MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 576

March 1980

META-RULES: REASONING ABOUT CONTROL

Randall Davis

ABSTRACT

How can we insure that knowledge embedded in a program is applied effectively? Traditionally the answer to this question has been sought in different problem solving paradigms and in different approaches to encoding and indexing knowledge. Each of these is useful with a certain variety of problem, but they all share a common problem: they become ineffective in the face of a sufficiently large knowledge base. How then can we make it possible for a system to continue to function in the face of a very large number of plausibly useful chunks of knowledge?

In response to this question we propose a framework for viewing issues of knowledge indexing and retrieval, a framework that includes what appears to be a useful perspective on the concept of a *strategy*. We view strategies as a means of controlling invocation in situations where traditional selection mechanisms become ineffective. We examine ways to effect such control, and describe *meta-rules*, a means of specifying strategies which offers a number of advantages. We consider at some length how and when it is useful to *reason about control*, and explore the advantages meta-rules offer for doing this.

Acknowledgements

This work was begun while the author was at Stanford University, supported in part by the Defense Advanced Research Projects Agency under DARPA Order 2494; by a Chaim Weizmann Postdoctoral Fellowship for Scientific Research, and by grant MCS 77-02712 from the National Science Foundation. The work has continued at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology with support by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract N00014-75-C-0643, and by the National Library of Medicine under Grant 1 PO1 LM 03374-01.

CONTENTS

1 INTRODUCTION

How can we insure that knowledge embedded in a program is applied effectively? Traditionally the answer to this question has been sought in different problem solving paradigms and in different approaches to encoding and indexing knowledge. Paradigms explored have included means-ends analysis, resolution, heuristic search, problem reduction, etc., while indexing and retrieval have been based on name (as in standard procedure invocation), effect (goal-directed invocation), and context (event-triggered processes). Each of these is useful with a certain variety of problem, but they all share a common problem: they become ineffective in the face of a sufficiently large knowledge base. Goal-directed invocation, for instance, offers a way of retrieving knowledge relevant to achieving a particular effect, but what happens if there are a hundred or more chunks of knowledge in a system that are capable of achieving that effect? In more general terms, how can we make it possible for a system to continue to function in the face of a very large number of plausibly useful chunks of knowledge?

In response to this question we propose a framework for viewing issues of knowledge indexing and retrieval, a framework that includes what appears to be a useful perspective on the concept of a *strategy*.

We view strategies initially as a means of controlling invocation in situations where traditional selection mechanisms become ineffective. We examine ways to effect such control and discuss a technique for implementing it that offers several advantages. We then consider how and when it is useful to reason about controlling invocation and describe a particularly convenient and simple mechanism for accomplishing such reasoning.

Sections 2 - 4 set the appropriate perspective for this work and give a brief overview of the basic ideas we have developed. Section 5 then describes our concept of a strategy and Section 6 describes the context in which it has been developed. Sections 7 and 8 discuss *meta-rules*, the means used to specify strategies, while Section 9 explores generalizations of the framework. These first eight sections offer a good general overview of meta-rules.

Sections 10 - 13 are an extended analysis of these ideas. Section 10 reviews briefly a number of other languages and problem-solving systems as a historical background. Section 11 discusses the problem of controlling invocation and examines the technique we have developed. Section 12 considers an interesting issue which arises is implementing invocation control, while Section 13 explores in detail our approach to reasoning about control. In each of these sections we examine the advantages our approach offers and consider what inherent limitations it possesses.

Section 14 considers what class of problems are appropriate for the machinery we have developed. Finally, Section 15 examines briefly another application of these ideas. Where the bulk of the paper is concerned with *controlling and reasoning about* invocation, Section 15 considers how the same set of ideas can be applied to *defining* invocation schemes.

2 PERSPECTIVE

Several points of perspective will help set this work in the appropriate context. First, this paper is concerned with issues of knowledge organization and general characteristics of knowledge representation, rather than the virtues of any specific encoding form (e.g., rules, procedures) or the utility of any particular control structure (e.g., backward chaining).

To encourage this view, we use the term *knowledge source* (KS) throughout, and mean it in the generic sense. It can be instantiated as procedure, theorem, inference rule, or other form of knowledge representation without substantial change to the existing text. Also, while they are called meta-*rules* for

historical reasons, the concept is more generally that of *meta-level knowledge*, or "knowledge about knowledge", and they could as well be meta-procedures, meta-theorems, etc. We refer to knowledge about a particular task domain as *object-level knowledge*; information about object-level knowledge is meta-level knowledge.

Meta-level knowledge is a sizable topic in itself (see, e.g., [11], [5], [52], [22]), too large to be treated in any depth here. Our interest in it in this paper is limited to the variety of meta-level knowledge concerned with control of invocation, even though our terminology is not always explicit. Occasional references are made to more general issues of meta-level knowledge, but the primary focus here is on the use of meta-level knowledge for control.

In addition, our perspective on meta-level knowledge is strongly pragmatic; philosophical issues are not the concern here. The point at hand is simply that there is both (object-level) information about a task domain and (meta-level) information about that information. Both are useful: much of the recent work in knowledge-based systems (e.g., [16], [30], [44]) has demonstrated the power in assembling large stores of object-level knowledge; in this paper we explore a framework for and the utility of assembling a body of meta-level knowledge.

A second point of perspective is suggested by Figure 1. For the purposes of this paper we view AI problem-solving programs as structured along the lines shown there. The *knowledge base* is the program's store of task-specific knowledge that makes possible high performance. The *inference engine* is an interpreter that uses the knowledge base to solve the problem at hand, repeatedly selecting KSs from the knowledge base and applying them.

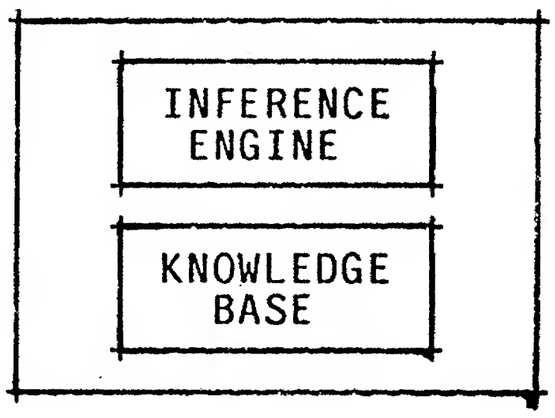


Figure 1 -- Simplified structure of a problem solving program

While this structure is clearly too simple to be an appropriate way to view every system, it seems broadly enough applicable to provide a useful point of reference for this discussion. It offers as well two specific advantages. First, it makes it easy to illustrate and explore the issues we wish to cover, and second, it provides some historical veracity: The rule-based system described in Section 6 resembles Figure 1, and provided the context in which meta-rules were developed. While the architecture of the system described there is very useful for purposes of illustration, no recommendation is being made that the rule-based knowledge representation or backward chaining control structure it uses is necessarily a good approach to problem solving.

The process of KS invocation is a central focus of this paper. For the purpose of discussion, we view it as occurring in three steps: retrieval, refinement, and execution. In *retrieval*, some KS property (name, pattern, goal, etc.) is used to select from the knowledge base a subset of KSs (e.g., "all those with the following goal descriptor..."). We term that subset the *set of plausibly useful KSs* (or PKS set), because they are the ones which may plausibly be applied to the current state of the problem.

During the *refinement* phase, the PKS set may be pruned or possibly (re-)ordered, to provide a finer degree of control over KS use. That is, having retrieved several KSs according

to the basic invocation criterion, we may also wish to make use of some advice about how best to use those KSs.

The final phase is *execution*, in which one (or more) of the KSs in the revised PKS set is applied to the problem.[2]

This paper focusses primarily on the refinement phase and considers issues of knowledge organization and use at that stage of the invocation process. We consider in particular three central topics: (a) the utility of refinement as a means of controlling invocation (Section 11), (b) the utility of a technique we call "content reference" as a way of implementing refinement (Section 12), and (c) how and when it is useful to reason about refinement, i.e., reason about control (Section 13).

In more detail, we suggest first that a certain style of invocation control (manipulation of the set of plausibly useful KSs) appears to be both a reasonably powerful and fairly natural way of controlling invocation and encoding strategy information. While there are many ways to implement such control, we suggest next that one particular reference technique (which we call content reference) offers a number of advantages, including flexibility of the resulting system. Finally, we show how the mechanism used to arrive at the appropriate reordering can be a general problem-solving and inference mechanism, and more specifically, the same inference mechanism in use at the object level.

Note also that we are not advocating meta-rules as a preferred language for strategies. While a rule-based representation does present some advantages for encoding reasonably small, modular chunks of knowledge (as we have noted elsewhere [9]), they are not particularly well suited to expressing larger, more complex constructs. The design of a good strategy language is still very much an open issue.

One final point: The mechanisms we have developed are designed for a uniprocessor and many of the arguments presented below assume availability of only a single cpu. A number of other approaches become possible (see, e.g., [46], [26], [29]) if multiple processors are available.

3 BACKGROUND -- THE PROGRAM AND THE PAPER

The work reported here was done as part of the development of TEIRESIAS [8], a program intended to assist in building task-oriented, knowledge-based consultation systems.[3] TEIRESIAS was developed in the context of the MYCIN system ([44], [10]) and it was in this environment that meta-rules were implemented. While MYCIN provided a valuable laboratory for exploring the ideas described in this paper, in some ways it also proved limiting (see, e.g., Section 11.3.5). The combined TEIRESIAS-MYCIN system design should, as a result, be viewed in historical perspective, rather than interpreted as a model to be adopted verbatim. Still, many of the insights gained from working on the system appear to be widely applicable; they form the central focus of this paper.

About a dozen meta-rules were completely specified over the course of TEIRESIAS's development. There are a number of reasons why relatively few meta-rules were developed. At the time meta-rules were created, the MYCIN system was still small enough that exhaustive invocation of its (object-level) rules was feasible, hence the meta-rule mechanism had somewhat

---

2. Like most such distinctions, this division into three steps is not absolute, since a finer degree of indexing during retrieval can substitute for some refinement. We argue in Section 5, however, that it is unwise to attempt to avoid all refinement with ever more detailed indexing schemes.

3. In what follows, we refer to the consultation system as the "performance program", to distinguish it from TEIRESIAS.

the character of a solution looking for a problem. Also at that time the primary goal of work on MYCIN was further specification of the basic medical knowledge used to get the correct diagnosis, rather than the somewhat more esoteric information used in finding a diagnosis efficiently. As a result, there was relatively little effort put into seeking out meta-rule examples in this domain.

Thus we cannot report on experience with a system so large that the guidance provided by meta-rules was a crucial addition. The examples are nonetheless real, in the sense that they were implemented and ran in a sizable knowledge-based system.

Some of the issues discussed here have appeared in print before, primarily in [8] and [7]. This paper offers both a reevaluation of those ideas and a number of new developments. It has the added benefits of hindsight and the opportunity to set this work in the context of systems which have been developed after meta-rules first appeared.

4 OVERVIEW OF IDEAS

The analysis and development that follows is somewhat detailed, yet the conclusions are reasonably straightforward. To insure that the main ideas are not lost in the detail, we present them here, in slightly oversimplified form. The first four ideas appear to have widespread applicability; the next four are somewhat more specific to our approach to doing refinement.

4.1 Saturation

We begin by noting that

> *Almost all traditional problem-solving control structures are susceptible to* **saturation,** *the situation in which so many applicable knowledge sources are retrieved that it is unrealistic to consider exhaustive, unguided invocation.*

Procedure invocation in traditional (i.e., ALGOL-like) programs is deterministic -- only one procedure is considered for invocation at any given moment. Many of the programming paradigms developed in AI, however (e.g., production rules, PLANNER-like languages), admit (or even encourage) the possibility of retrieving several chunks of knowledge, all of which are plausibly useful in a single situation. Typically, in these paradigms, the KSs are retrieved unordered and are invoked exhaustively, each considered in turn, until some stopping criterion is met or until all have been tried. However, faced with a large enough or varied enough set of alternatives, blind, exhaustive invocation becomes infeasible.

One useful approach to controlling saturation is by **refining** *the set of KSs retrieved.*

That is, we might prune and/or reorder the set. This approach to controlling the use of knowledge offers significant leverage. Useful gains in performance can result from adding to a system a store of (meta-level) knowledge which indicates which chunk of object-level knowledge to invoke next. For example, if the set of KSs retrieved can be pruned, we may be left with few enough that all can be attempted; if the set can be appropriately reordered, then we may be able to solve the problem at hand with one of the first KSs invoked.

We study in some detail the way the refinement process might be carried out. We propose a mechanism based on the use of meta-rules and compare it to other techniques which have been used to provide a similar capability (e.g., PLANNER's recommendation list, the notion of conflict resolution in production systems, etc.). Our approach appears to be a generalization of other techniques, in part because decisions about refinement are made using a fairly powerful inference mechanism.

4.2 Reasoning about control

Determining the correct refinement (i.e., pruning and reordering) of the set of KSs retrieved can be viewed as a problem solving task.

That is, rather than attempting to deal with refinement on an ad hoc or case by case basis, we suggest a broader perspective that considers it as a full-fledged problem to be solved using the repetoire of AI problem solving techniques

We then suggest attacking this problem with a general inference mechanism, one which will allow the program itself to reason about and deduce the appropriate refinement to use. This reasoning is also done at the time that refinement is required, and hence is dynamic and responsive to the current state of the problem. This appears to be more powerful and flexible than attempting to anticipate the problem *a priori* and building in fixed strategies.

Since the phrase "reasoning about control" is somewhat provocative, we go to some lengths in Section 13 to explain what we mean by this and to demonstrate in what sense TEIRESIAS does in fact reason about control.

4.3 A unified reasoning mechanism

The same inference engine used to reason about problems at the object level can be employed to reason about control at the meta level.

We suggested above that we can treat reasoning about control as a problem solving task. This does not, however, mean that a whole new problem solving mechanism needs to be built to deal with it. We describe here a system in which exactly the same mechanism is used to reason about both object-level tasks (the original problem) and about meta-level tasks (control of invocation).

The reasoning mechanism used here has a traditional knowledge-based system architecture. In applying it at the meta level, we are tackling an interesting and novel task domain: refinement, or more generally, control of invocation. As a result, an important part of the effort involves assembling a body of knowledge and heuristics about guiding invocation.

We will see in Section 12, by the way, that this same inference mechanism has yet another application: It is used to reason about the content of rules.

4.4 Explicit representation of control

The knowledge used in doing refinement should be represented explicitly, and should be encoded in constructs that are accessible to the program itself.

This is in part a necessary precondition to some of the ideas above. In order to reason about control, it is necessary to have access to it. Yet this is not often done: Such information is often embedded in the code of the interpreter, making it both implicit and inaccessible.

Explicit representation of this information offers benefits in terms of producing a system whose behavior is more transparent and which is less prone to the class of bugs described in Section 11.2.1.

Representing this information in constructs accessible to the program makes possible a hierarchy of strategies, which in turn offers both a useful framework for organizing knowledge and an additional mechanism with which to guide program performance. Having such a hierarchy also means that strategy knowledge in the system is at once both active inferential knowledge used to control invocation and data which is examined by other parts of the system.

4.5 Connections

Several of the ideas mentioned above have broader connections to other work in AI.

For example, an interesting theme of some recent research is the explicit, accessible representation of various forms of knowledge which were traditionally left implicit. Work in the past several years has explored the utility of explicit representation of: plans [41], grounds for assertions [15], tasks [33], data structure architecture [12], and information about the internal state of the problem solver ([34], [14]). Here we are proposing a method for explicit representation of strategic and control knowledge.

In each case the knowledge represented offers the corresponding program access to a part of itself that was previously inaccessible. Having gained access, a number of things become possible. In [12], for example, the explicit representation of data structure architecture was used to support knowledge acquisition. In several systems, access to new information was used to support reasoning about the information represented. The work in [41], for instance, produced a more powerful problem solver capable of reasoning about the structure of its plans; the work in [49] showed how reasoning about grounds for assertions makes possible dependency directed backtracking.

Here we explore how reasoning about control can support useful problem solving behavior. We will also see (Section 12) that TEIRESIAS is capable of a simple form of reasoning about the content of its object-level knowledge.

A second theme that has received attention in recent AI work is the concept of explicit control of invocation. In TEIRESIAS this was motivated by the desire to control potentially explosive search, and is instantiated as a mechanism for guiding a program by careful refinement of the PKS set. Work on AMORD [14] has also emphasized the importance of explicit control, motivated by early experience with PLANNER. Where PLANNER's chronological backtracking was essentially unguided and kept control buried in the interpreter, AMORD provides a mechanism (data dependencies) for informed, explicit control of backtracking.

A third theme, implied by the ideas above, is the utility of separating what is known about solving a problem from the decisions about how to use that knowledge. In our formulation, this corresponds (roughly) to the distinction between object-level knowledge (information about objects in the domain) and meta-level knowledge (information about how to use the object-level knowledge). Given our desire to reason about control, such a distinction appears necessary. It also appears to be quite useful, since, as discussed in Section 12.4.1, it offers advantages in terms of flexibility of the resulting system.

Similar distinctions have been proposed by a number of authors for a variety of motivations. Hayes [22] has emphasized the difference in the kinds of information involved and the clarity that arises from expressing each distinctly. Kowalski [27] has claimed that keeping them separate aids in writing correct programs and verifying them. Pratt [38] has speculated that the distinction might be the basis for a new sort of high level language, in which the program itself takes over "efficiency management" in much the same way that current languages have taken over storage management.

Our goals are somewhat less ambitious. We find that the distinction makes possible multiple uses of knowledge (see [11] and Section 12.3), makes possible the direct statement of control information (Section 11.2.1), and may contribute to the ease of system construction (Section 15).[4]

---

4. Note also that the distinctions we have drawn (what is known vs how to use it, and object-level vs meta-level knowledge) may be somewhat artificial. We would not attempt to justify them on rigorous epistemological grounds. Yet as a conceptual aid they appear quite useful: It seems much easier in practice to specify the desired (meta-level) knowledge if we proceed *as if* the distinctions were clearcut.

To review briefly: we have suggested that computation can be guided by refinement, that refinement can be accomplished via reasoning, and that a single reasoning mechanism can be employed at all levels. The next several themes examine the basic idea of refinement and its value as a mechanism for attacking saturation.

4.6 The concept of a strategy

We define a strategy to be knowledge about which KS to use when more than one is applicable.

Faced with the necessity of choosing which KS to invoke next from among all those retrieved, we require some guidance. We define the notion of a strategy to be that guidance.

We then generalize this concept slightly and view strategies as one form of meta-level knowledge (see [11] for a review of some other types). They are seen as information about the use of object-level knowledge and function as a means of controlling invocation by guiding the selection of object level knowledge.

Two points of terminology are important here. First, we use the terms refinement, invocation control, strategy, and meta-rule interchangeably, and mean by them advice about pruning or reordering the set of KSs retrieved. Second, we are using the concept of meta-rules in the restricted sense of referring to rules about control. Just as there are several forms of meta-level knowledge, there are correspondingly several forms of meta-rules. In this paper, however, we concentrate on meta-rules about control and use the term in that sense alone.

4.7 Organization of strategies

Strategies can be associated with the same KS properties (e.g., goals) which are used to do retrieval.

Indexing and retrieval of object-level knowledge has been done using a range of different properties (names, effects (goals), context (events), etc.). It is useful to employ those same indexing points for strategies. This technique makes it possible to encode strategies that are local and specific to a given goal (and hence potentially quite effective), yet those strategies will not impose any overhead when the system is working on other goals in the problem. This technique also provides an interesting perspective on organizing and using knowledge to control search: In addition to writing sophisticated search algorithms, we can associate search guidance information with the nodes of the search space itself.

4.8 The utility of content reference

As previous work in problem solving languages judged the traditional invocation by name mechanism inadequate at the object level, so it appears equally inadequate for use in strategies at the meta level.

As we explore below, non-deterministic reference mechansims prove very useful at the meta level. We find, however, that standard non-deterministic mechanisms (e.g., pattern-directed invocation) are insufficiently powerful. We propose instead a different facility:

A technique we call **content reference** *offers a generalization of standard non-deterministic invocation mechanisms.*

As we will see, this technique is useful in implementing the pruning and reordering effected by strategies. It leads to a system with a number of interesting characteristics, including a high level of flexibility in the face of changes to the knowledge base.

4.9 A generalized view of invocation

Viewing invocation in terms of retrieval, refinement, and execution leads to a more general perspective on what it means to invoke a KS.

As described in Section 15, this view encourages a broader perspective on the styles of invocation we might have, by emphasizing that the indexing and retrieval of a KS can be based on wide variety of properties (not just goals, data, etc.). Those properties are best thought of as arbitrary *descriptions* of a KS, rather than *names* or pointers to it (as in standard procedure invocation). This makes possible a wider range of invocation schemes.

All of these ideas are elaborated and illustrated in the sections that follow.

5 STRATEGIES: GUIDING THE USE OF KNOWLEDGE

There have been two fundamentally different kinds of responses to the problem of saturation. The first approach suggests that new accessing, indexing and knowledge organizing schemes need to be developed, to provide a more sharply focussed degree of KS retrieval. The progression from the venerable British Museum algorithm (exhaustive, breadth-first forward search) to means-ends analysis was motivated in part by just such considerations (see [17] and [19]). The belief there was that if applying all operators relevant to the current state leads to explosive growth of the search space, then perhaps by computing the difference between the current and goal states, and retrieving operators relevant to that difference, fewer operators will be retrieved and invocation will be more focussed.

A more recent example is the notion of procedural attachment in the work on frames. This too, provides a means of organizing, indexing, and accessing operators which tries to insure that retrieval is sharply focussed. Relevant knowledge is clearly recognizable and easily retrieved, because few procedures are attached to a slot and each procedure is labelled with a particular purpose ("when needed", etc.).

A second basic response to the saturation problem --- and the one we examine --- is to accept the existence of saturation and to provide a mechanism for guiding the system in spite of it. That is, given that there are likely to be too many KSs retrieved no matter what indexing or organizational scheme is used, can we supply the program with a means of selecting from or perhaps reordering the KSs so as to make the best use of whatever computational resources are available?

We have taken this as a starting point, and offer our initial definition of strategy in terms of it. We suggest that a strategy can profitably be viewed as *knowledge about which chunk of knowledge to invoke next when more than one chunk is applicable.* It is our contention that this is an important site for the embedding of knowledge, because system performance will be strongly influenced by the intelligence with which the decision is made. We claim also that it is a decision which in many systems is made on the basis of knowledge that is neither explicit, nor well organized, nor considered in appropriate generality.

The utility of this approach to dealing with saturation arises in part from the apparent inevitability of the phenomenon. Almost by definition, AI problems are to some degree ill structured (in the sense defined in [37] and [45]). This means there will be some degree of non-determinism in solving them and hence at least the possibility of saturation. To see this, consider the character of the complementary set of problems, the well structured problems. One

of the defining characteristics of such problems is that they have algorithmic solutions and can thus be solved by the use of a predetermined sequence of operators. At each point of the solution process there is no question about which KS to invoke next and no doubt that we are getting closer to the solution. For ill structured problems, on the other hand, the choice of which KS to use next is at best an informed guess from among several possibilities and there is no guarantee that we are indeed making progress toward a solution. When there are several possibilities, we are vulnerable to saturation if the knowledge base grows sufficiently large.

Thus, the inherently ill structured nature of AI problems is one important potential source of saturation. In addition, recent empirical experience in the field suggests that powerful programs require large stores of knowledge. The larger a knowledge base, the more likely it is that multiple KSs will be retrieved at each invocation cycle, giving rise to a second important source of saturation.

Finally, as McDermott has noted ([33]), recent experience in knowledge engineering makes it appear unlikely that retrieval of multiple KSs can be avoided by supplying increasingly more specific preconditions (i.e., trying to substitute a finer degree of retrieval for refinement). Since the knowledge base for a large system is built incrementally, additivity is an important characteristic. In particular, adding the Nth KS to a system should not require comparing it in detail with all N-1 other KSs. Yet this is exactly what we would have to do to insure that its preconditions (and hence retrieval) were specific enough that the number of KSs retrieved at any one time was small. Such comparisons are at least difficult and likely to be unsuccessful in the long run. They encourage writing ever larger preconditions tailored precisely to the current state of the knowledge base and current subset of problems under study. Changes to either of these may necessitate revising what is now a set of strongly *inter*dependent preconditions. It thus seems impractical to substitute ever more precise retrieval for all refinement.

As a result, it appears that saturation is (at least currently) almost inevitable, no matter how clever our indexing and retrieval schemes. This argues for the utility of some sort of invocation guidance mechanism: By allowing retrieval of multiple KSs and adding a mechanism for evaluating those retrieved, we ease considerably the task of system construction.

One final point. While the evidence for inevitable saturation in large systems seems compelling, it should be noted that much of what we propose does not depend on saturation and is more generally applicable. The mechanism for guidance that we discuss can be useful even when a program is not immobilized by saturation. In that case, strategies can supply an additional degree of focussed behavior, possibly allowing a program to solve a problem faster than it might have otherwise.

6 ARCHITECTURE OF THE PERFORMANCE PROGRAM

Earlier we distinguished between TEIRESIAS and the performance program. This section describes the architecture of the performance program as background for further discussion.

We assume that the performance program is structured along the lines suggested in Figure 1, with the knowledge base and inference engine described earlier. The knowledge base is assumed to contain inference rules of the form shown in Figure 2 below; both the internal (i.e., LISP) and English forms are shown. For purposes of illustration we deal with a body of knowledge about selecting stock market investments; the performance program thus functions as an investment advisor.[5]

Each rule specifies an action (in this case a conclusion) which is valid if the Boolean combination of preconditions given in the premise has been met. Both preconditions and actions are stated in terms of functions on associative triples; e.g., the first precondition here is "timescale (*attribute*) of investment (*object*) is long-term (*value*)". The rules are judgmental, i.e.,

they can specify inexact inferences, with the strength of the inference given on a scale from 0 to 1. Conclusions made by two rules are combined using the model of inexact reasoning described in [43]; the details of that model will not concern us here.

```
RULE027
If   [1] the time scale of the investment is long-term,
     [2] the risk level of the investment should be low risk,
     [3] the economy is headed for a recession,
then Pacific Gas and Electric is a likely (.7) choice for the investment.

PREMISE     ($AND (SAME OBJCT TIMESCALE LONG-TERM)
                  (SAME OBJCT RISK-LEVEL LOW-RISK)
                  (SAME OBJCT RECESSIONONTHEWAY TRUE))

ACTION      (CONCLUDE OBJCT STOCK-NAME PGE .7)
```

Figure 2 -- example of an object-level rule

The rules are invoked in a goal-directed backward chaining mode that produces an exhaustive depth-first search of an and/or goal tree. Suppose, for instance, that the system is attempting to deduce which stock to invest in. It retrieves the (precomputed) list of all rules which mention STOCK-NAME in their action, and attempts to invoke each one in turn, evaluating their premises to see if the indicated conditions have been met. For the rule shown above, this means determining first whether the timescale of the investment is long-term. This is in turn set up as a subgoal and the process recurs.

The resulting search is depth-first, since each precondition is considered in turn. The tree is an and/or goal tree since premises may have disjunctions in them. The search is exhaustive because the rules are inexact (unless an answer has been deduced with certainty, it appears appropriate to continue to collect all evidence about a given subgoal).

The ability to use an exhaustive search is of course a luxury and can be computationally infeasible if the number of rules about a particular goal is large enough. In that case some choice would have to be made about which of the plausibly useful rules should be invoked. Note that here standard goal-directed invocation is insufficient. That is, it is of limited utility to know simply that a rule accomplishes a particular goal. We need a way of expressing a more varied set of properties about rules and a way of using those properties to "tune" the goal-directed invocation. *Meta-rules* were created to address this problem.

7 META-RULES -- FORM AND CONTENT

Two examples of meta-rules are shown below, in both the internal format and the English translation that results.[6] The first of them says, in effect, that in trying to determine the best investment during a time when the index of leading economic indicators (LEI) has been steadily climbing, rules that base their recommendations on the possibility of a recession are not

---

5. This stock market system is modelled after the MYCIN system. We have abstracted out here just the essential elements of MYCIN's design, and shifted the domain from medicine to the stock market via a few word substitutions (e.g., *E.coli* became *AT&T*, *infection* became *investment*, etc.). This shift was done to keep the discussion phrased in terms familiar to a wide range of readers, and to emphasize that neither the problems attacked nor the solutions suggested are restricted to a particular domain of application.

6. This work did not focus on natural language, and used the LISP-to-English translation facilities described in [42]. One weakness of that approach is evident here, as these particular rules suffer especially from the line-by-line approach to translation.

likely to be the right approach to take. The second indicates that when dealing with clients nearing retirement age, more secure stocks should be considered before more speculative ones.

```
METARULE001
If  [1] the LEI index has been climbing steadily for the past several months,
    [2] there are rules which mention in their premise that the economy may
        be headed for a recession,
then it is likely (.7) that each of these rules is not going to be useful.

PREMISE
 ($AND (SAME OBJCT LEI RISING)
        (THEREARE RULES (MENTIONS OBJCT PREMISE RECESSIONONTHEWAY) SET1))

ACTION        (CONCLUDE SET1 UTILITY NO .7)
```

```
METARULE002
If  [1] the age of the client is greater than 60,
    [2] there are rules which mention low risk,
    [3] there are rules which mention high risk,
then it is very likely (.8) that the former should be used before
the latter.

PREMISE
   ($AND (GREATER OBJCT AGE 60)
          (THEREARE RULES (MENTIONS OBJCT EITHER LOW-RISK) SET1)
          (THEREARE RULES (MENTIONS OBJCT EITHER HI-RISK)  SET2))

ACTION        (CONCLUDE SET1 BEFORE SET2 .8)
```

Figure 3 -- Two meta-rules

7.1 Rule content

There are several points to note about the character of the information conveyed by meta-rules. First note that we have rules which are making conclusions about other rules. That is, where object-level rules conclude about the stock market domain, meta-rules conclude about object-level rules. It is this "knowledge about knowledge" characteristic of their content (rather than any detail of their syntax) which makes them meta-rules.

Second, the meta-rules shown make use of the model of inexact reasoning developed for object-level rules. But the two ideas --- meta-level knowledge and inexact reasoning --- are quite independent, and meta-rules could have employed standard binary logic. The combination does, however, offer several interesting advantages, as we discuss in Section 11.2.

Third, the conclusion of Meta-rule 1 concerns the *utility* of some object level rules, not their validity. That is, Meta-rule 1 does not indicate circumstances under which some of the object level rules are invalid (or even "very likely (.9)" invalid). It merely says that they are not likely to be useful; i.e., they will probably fail, perhaps only after requiring extensive computation to evaluate their preconditions.

As an example, consider Meta-rule 1. If the LEI index has been rising recently, Meta-rule 1 will indicate that Rule 27 is not likely to be useful. Rule 27 will as a result be pushed toward the end of the list of rules to be used (details of this mechanism are given in

Section 8). This will delay use of Rule 27 until other, more promising rules have been tried.

Such delaying tactics are useful because a recession may require a significant amount of computation to detect, yet is unlikely (but not impossible) if the LEI index has in fact been rising. If invocation of object-level rules is not exhaustive, delaying Rule 27 means available computational effort will be devoted to those rules which seem more likely to be productive. If invocation is exhaustive, reordering the rules is still useful, since the program's performance will appear more rational if it tries what seem to be more appropriate lines of reasoning first. It may also discover that one of those more appropriate lines of reasoning solves the problem, saving the significant amount of work required to attempt diagnosis a recession.

Strategies of the sort embodied in Meta-rule 1 are common in other domains as well. In medicine, for example, a ten year old child complaining of sudden, sharp abdominal pains, nausea and vomiting might in fact be suffering from a rather rare metabolic disorder called acute intermittent porphyrea. But the disease is rare in children (though not definitively ruled out by age alone) and enough work to diagnose that "it's one of the last things to think of" for someone that young. It might be in fact be the problem, but a physician would use the (meta-)rule of thumb that porphyrea is unlikely enough that he can defer the work necessary to diagnose it until other possibilities (e.g., appendicitis) have been explored.

Note also that the information about *utility* represented in meta-rules is very different from the information about *validity* represented by object-level rule preconditions, and the two cannot be mixed. If, for example, we had tried to represent the information about a rising LEI index by making it a precondition of Rule 27 (i.e., adding the clause "`the index of LEI has not been rising for the past several months`"), then the rule would be guaranteed to fail (and we could never detect a recession when the LEI index was rising). Similarly, if we added a clause about age to the rule(s) diagnosing porphyrea we could never deduce the presence of the disease in children. But this is not what we want. We simply want to capture the notion of more appropriate or less appropriate lines of reasoning, and delay (not throw out) the less appropriate ones.

The information in meta-rules is thus advice about the likely utility of object-level rules. It can help guide program performance but it is different in kind from the information carried in an object-level rule premise.

7.2 Rule syntax

In the current implementation the conclusions made by meta-rules can be of two forms. As in the first meta-rule, they can make deductions about the likely utility of certain object level rules, or (as in the second) they can indicate a partial ordering between two subsets of object level rules.

The syntax for meta-rules is identical to object-level rule syntax, extended to include a few new predicate functions (e.g., `MENTIONS`, and its complement, `DOESNTMENTION`), and the two new attributes noted above (a rule's `UTILITY` and its place in the sequence of rules to be invoked, `BEFORE`). Two important benefits accrue from using a single syntax. First, the system has a uniform encoding of all levels of knowledge, and second, meta-rules may employ all the existing machinery of certainty factors to make inexact statements. Implications of both of these are discussed below.

While the syntax of the current implementation is simple, it offers a useful range of expression. In describing the utility of object-level rules, meta-rules can indicate that

```
under conditions A and B,
                                        at all
rules which do {not} mention X          in their premise
                                        in their action
will  definitely be useless.
      probably be useless.
       ...
      probably be especially useful.
      definitely be especially useful.
```

In describing a partial ordering of the object-level rules, meta-rules can indicate that

```
under conditions A and B,

                                        at all
rules which do {not} mention X          in their premise
                                        in their action

should  definitely      be used         first.
        probably                        last.
                                        before
         ...
        possibly                        after

                                        at all.
rules which do {not} mention X          in their premise.
                                        in their action.
```

The current syntax also permits conjunctions and disjunctions of phrases specifying "`rules which do {not} mention...`". If, for example, it became necessary to make Meta-rule 1 more precise about the kind of object-level rules it referred to, then we might replace its second clause with a conjunction of the form "`rules which mention recession and which do not mention short term interest rates.`"

Note that the primitives that have been added to the syntax deal with characteristics of knowledge. For example, the new predicate functions are based on the fact that the knowledge representation is a composite structure whose components are accessible; hence, it makes sense to refer to the content of a rule. The new attributes are based on the recognition, first, that a rule invocation is a discrete event, so it makes sense to indicate order; and, second, that one rule may be more useful than another, so we can talk meaningfully about utility.

These simple additions allow the statement of the range of strategies indicated above and have so far met current needs.

8 META-RULES -- FUNCTION

We present first a simplified picture of meta-rule function, then elaborate the details below. Adding meta-rules to the system requires only a minor addition to the control structure described above. As before, the system retrieves the list (call it L) of all rules relevant to the current goal. But before attempting to invoke them, it first determines whether there are any meta-rules relevant to that goal.[7] If so, these are invoked first, using the standard invocation mechanism originally designed for object-level rules. As a result of their action, we may obtain a number of conclusions about the likely utility and desired relative ordering of the rules in L.

---

7. That is, are there meta-rules directly associated with that goal. Meta-rules can also be associated with other objects in the system; the range and utility of such indexing points is a sizable topic considered in more detail in [8].

These conclusions are used to reorder or shorten L (details of this are given in the next section), and the revised list of rules is then passed to the rule interpreter described in Section 6. Viewed in tree-search terms, the current implementation of meta-rules can either reorder the branches of the tree or prune them (pruning occurs if it is possible to conclude with certainty that some rule will not be useful).

Since there is no reason to have only two levels of control, the current implementation supports an arbitrary number of levels, each serving to direct the use of knowledge at the next lower level. The system retrieves the list (L) of object level rules relevant to the current goal. Before invoking this, it checks for a list (L') of first order meta-rules which can be used to reorder or prune L. But before invoking this, it checks for second order meta-rules which can be used to reorder or prune L', etc. Recursion stops when there is no rule set of the next higher order, and the process unwinds, each level of meta-rules advising on the use of the next lower level.[8]

8.1 Details

To examine meta-rule function in more detail, we consider the invocation of the two rules shown earlier in Figure 3.

Assume the system is attempting to determine which stock to invest in. It will retrieve the list of all object-level rules concluding about stock name (call the list L) and then the list of meta-rules associated with stock name. Assume the process ends here because there are no second-order meta-rules and assume that the first meta-rule in the list is METARULE001. If the LEI index has been rising, the first clause succeeds. Evaluation of the second clause

```
(THEREARE RULES (MENTIONS OBJCT PREMISE RECESSIONONTHEWAY) SET1))
```

results in assigning to SET1 the subset of rules that mention in their premise the concept that a recession may be imminent.

Determining which of the object-level rules do in fact mention this attribute is accomplished by direct examination of the code of the object-level rules. That is, each of the rules in L is tested by the function MENTIONS, which examines the source code of the rule to see (in this case) if that rule mentions the attribute RECESSIONONTHEWAY in its PREMISE. TEIRESIAS has two ways of carrying out this examination. The simpler method is based on the use of a template associated with each of the predicate functions and is described here. We defer description of the more complex method to Section 12.3.

The template associated with a predicate function describes the format of a call of that predicate function, by giving the generic type and order of arguments to the function. It thus resembles a simplified procedure declaration (Figure 4).

Function	Template	Sample function call
SAME	(SAME OBJCT ATTRIB VALUE)	(SAME OBJCT TIMESCALE LONG-TERM)

Figure 4 -- Example of a function template

---

8. No examples of second level or higher meta-rules were discovered in the current domain, but relatively little effort was devoted to discovering them. Also recall the caution in Section 4.5: We consider the notion of a hierarchy of rules to be simply a pragmatic tool useful in explicating the relevant knowledge, rather than an emphatic statement of epistemology.

The template is not itself a piece of code but is simply a list structure of the sort shown above, indicating the appearance of an interpreted call to the predicate function. Since rules are kept in source code form, the template can be used as a guide to dissect a rule. For each clause in a rule, TEIRESIAS retrieves the template associated with the predicate function found in that clause (i.e., the template associated with the CAR of the clause), and uses it to guide the examination of the clause. In the case of the function SAME, for instance, the template indicates that the attribute (ATTRIB) is the third element of the list structure that comprises the function call. Thus to see if a rule mentions the attribute *possibility that the economy is heading for a recession*, the system uses the appropriate template to extract the attribute from each clause and see if it is "RECESSIONONTHEWAY".

Note that one part of the system is examining the code (the rules) being executed by another part. Note also that this examination is guided by the information carried in components of the rules themselves. The ability to examine the code could have been accomplished by requiring all predicate functions to use the same format, but this is obviously awkward. Allowing each function to describe the format of its own calls permits code to be stylized without being constrained to a single form, and hence is flexible and much easier to use. This approach requires only that each form be expressible in a template built from the current set of conceptual primitives. It also insures that the capability will persist in the face of future additions to the system.

Implications of and more information about this direct examination of source code --- which we call *content reference* --- are explored in Section 12.3.

Having determined which object-level rules mentions the possibility that "the economy may be heading for a recession", we have completed evaluating the premise. The action part of METARULE001 then concludes that each rule meeting this test is not likely to be useful.

Evaluating METARULE002 proceeds analogously: If the client is more than 60 years old, SET1 is assigned the list of all rules mentioning low-risk in either their premise or action, SET2 is assigned all rules mentioning high-risk. The conclusion would then indicate that it is likely that each rule in SET1 should be done before any of those in SET2.

When all the meta-rules have been applied, they will have made a number of conclusions about the utility and relative order of the rules in L. The task now is to sort and perhaps prune L, based on those conclusions. Since the transitivity of the order relation often introduces constraints that are not explicitly mentioned by a meta-rule, it is first necessary to compute the transitive closure of the set of ordering constraints.[9] A straightforward implementation of Warshall's algorithm [51] supplies this.

The list L will be pruned if there are any conclusions which indicate that some rules will definitely be useless, and hence can be deleted. For the remainder, the most useful rules should be tried first.

The final step is thus a sort-and-delete pass through L using the following criteria:

```
If the utility of a rule is -1 (meaning "definitely not useful"),
        delete it from L,
otherwise
        rule X goes before rule Y if
                it is required by ordering constraints, or
                the utility of X is higher than the utility of Y
```

---

9. For instance, if "do X before Y" and "do Y before Z" are indicated by rules, often there is no rule that indicates the necessary condition of "do X before Z." (The system does not currently check for an inconsistent set of orderings, but should someday be upgraded to do so.)

The result is a reordered and possibly shortened list.

The entire process is summarized below (simplified by assuming that there is only one level of meta-rules):

```
To deduce the value of an attribute A:
  1) set L to the list of rules which conclude about A
  2) set L' to the list of meta-rules associated with A
  3) Evaluate each of the rules in L'; this may result in some conclusions
     about the rules in L
  4) Sort and prune L according to the criteria shown above
  5) Evaluate each of the rules in L; this may result in conclusions about
     the value of A
```

9 GENERALIZATIONS OF OUR APPROACH

There are several simple generalizations of the approach described above which will help to make clear the range of applicability of our knowledge organization framework. We consider both several types of meta-rules and generalizations of the use of meta-rules.

9.1 Varieties of meta-rules

In addition to viewing them in general terms as meta-level knowledge, meta-rules may be classified by a detailed breakdown indicating the kinds of uses of which they can be put. One use, illustrated by the meta-rules above, might be labelled "invocation utility", since it is concerned with the likely benefit to be obtained by invoking a class of object-level rules. The rules about recession, for example, have low invocation utility during a period of rising LEI index, because (as noted) they involve a lot of work, yet are unlikely to come up with an answer.

A second use of meta-rules is their application to the problem of subgoal ordering and interactions. While MYCIN did not prove to be an appropriate system in which to explore this use, other systems have developed mechanisms similar to meta-rules and applied them to the interaction problem. In [33], for example, "choice rules" are used to handle interacting subgoals. In that system, the design of an electrical circuit (e.g., a passive band-pass filter) is decomposed into a number of "mostly independent" subgoals (e.g., a low-pass filter and a high-pass filter). Each subgoal is worked on in turn, but potential interactions are handled by choice rules. For example, given several ways to build the high-pass filter, one particular class of methods (which result in low output impedance) would be favored because the filter is to be cascaded with another (low pass) filter in the circuit. Thus it is the presence of other, interacting goals, which suggests the reordering or pruning of the KSs working on this goal.

Finally, a third application of meta-rules might be called "spectator satisfaction" rules. In some domains (e.g., medicine), the collection of information and consideration of subgoals follows a stylized order. This order, while not necessarily important for problem solving performance, is useful to have in the system since it adds a degree of familiarity to system behavior, and can contribute to human engineering considerations. Rules similar to Meta-rule 2 are useful for representing this form of information.

9.2 Generalizing the use of meta-rules

The discussion so far has dealt with the use of rules guide the invocation of rules. This need not be the case, since a wide range of knowledge representations could be used at any level. We might have procedures guiding the use of inference rules (or vice versa), procedures guiding procedure invocation, etc. While there are substantial advantages to keeping the representation uniform at all levels (see Section 11.1), this is not a basic limitation of the approach. The fundamental idea is simply that of building a store of (meta-level) knowledge to guide the use of (object-level) knowledge; "meta-procedures" or "meta-theorems" would do as well.

As a second generalization, consider that meta-rules can be used to tune a variety of control structures, in addition to goal-directed invocation. They can be used, for instance, to guide data-directed invocation as well, and this has been done. Before execution of the set of (antecedent-style) rules triggered by making a conclusion, TEIRESIAS checks for meta-rules associated with that conclusion. These meta-rules prune or reorder the set of antecedent rules to be used. This makes it possible to control the depth and breadth of the actions triggered by the addition of new information to the data base.

Meta-rules might also be used in a system based on means-ends analysis. Given a difference to be reduced and a number of operators relevant to that difference, we might have meta-rules associated with each difference, offering advice on how best to order the relevant operators under varying conditions.

In the most general terms, the idea of "tuning" appears to be applicable to any control structure in which multiple KSs may be retrieved at once. The set of plausibly useful KSs will be determined by the control structure in use, be it goal-directed, data-directed, "difference-directed", or any other style of retrieval. A store of meta-level knowledge can then be used to "fine tune" that set by reordering or pruning it.

10 ANALYSIS

In this and the following three sections we analyze the strengths and limitations of the framework described above, and consider what new ideas it may have to contribute to organization of knowledge or problem solving. We do this by considering the three general themes mentioned in the introduction, and evaluate the meta-rule framework (a) as a mechanism for controlling and guiding invocation via refinement, (b) as a mechanism for implementing refinement, and (c) as a mechanism for reasoning about refinement. Each of these is considered from the perspective of (*i*) the utility of the basic concept, (*ii*) what aspects of our approach may be novel and the practical benefits that arise from those aspects, and (*iii*) the limitations and shortcomings in our current implementation.

10.1 Strategy encoding mechanisms

To provide a background for this analysis, we review a number of programming languages and problem-solving systems with respect to the strategy encoding mechanisms they supply or are capable of supporting. The basic question we ask of each is *What features of the language or system are of use in organizing, encoding, and using information to control invocation via refinement?*

We consider standard procedure (subroutine) invocation, GPS [17], traditional production rule systems [35], STRIPS [19], QA3 [20], HEARSAY-II [16], microPLANNER ([24], [47]), CONNIVER ([32], [48]), QA4 [40], and NASL [33]. One caveat: This comparison conveniently ignores the fact that each of these systems and languages had its own set of motivations and design criteria. We are not arguing that any of them should have been designed differently, but are instead simply using

them as points of comparison and illustration.

Procedure invocation represents a degenerate case, since all the selection is done ahead of time by the programmer and is "hardwired" into the code. That is, the programmer decides as he writes the code what the purpose of each procedure should be, which procedures should be invoked at each point, and which transfers of control should occur. As result there is no nondeterminism at runtime, no opportunity for choice, and hence no reason for refinement.

There is the potential for non-determinism (and hence opportunity for choice) in GPS, since a given difference may have several operators associated with it in the table of connections. However, no use has been made of this. The system used several varieties of exhaustive search but did not explore the effects of ordering the operators.[10]

Traditional production rule systems were one of the earliest programming technologies to pay explicit attention to issues of nondeterminism. The notion of *conflict set*, for instance, refers to the set of all rules whose preconditions for execution are currently satisfied and hence could plausibly be executed on the next cycle. To handle nondeterminism, the basic interpreter is augmented with selection criteria. A number of such criteria have been developed, ranging from a predefined priority ordering on the rule set, to more complex strategies that give preference to the rule least recently used, the most general rule, etc. (See [35] and [9] for a review of production rules and a more detailed discussion of conflict resolution strategies).

The STRIPS system is similar, in that it uses a single evaluation function embedded in the interpreter. The set of all operators relevant to a particular difference is ordered on the basis of estimated effort to achieve the preconditions, measured as the number of literals appearing in the precondition. It thus tries first the operator that apparently requires the least work to invoke.[11]

Applications of the QA3 language explored the use of some standard theorem proving strategies (set of support, etc.) in problem solving. The strategies acted to prune the set of clauses which might be resolved and hence were in effect global strategies which modified the basic inference engine. The system was novel in attempting to represent those strategies in the same language used to represent facts about the domain (predicate calculus).

The HEARSAY-II system illustrates a number of similar facilities in an event-driven system. In particular, the "focus of attention" mechanism is a single, global evaluation function used to select a KS from among those ready to fire. The function takes into account a rather wide range of factors and performs subproblem selection as well as operator selection. It is also somewhat more accessible, since it allows input parameters to be easily modified, giving the user the ability to explore a variety of approaches to attention focussing. The intent, here, however, is still to effect a single, global evaluation function based on a number of fairly general principles (see [23] for details).

PLANNER's pattern-directed invocation provides a facility at the programming language level for nondeterministic KS retrieval and its "recommendation list" offers a mechanism for encoding selection information. The "use" (THUSE) construct (Figure 5) provides a way of specifying by name which KSs (theorems) to try in which order. The theorem base filter (THTBF) construct offers a way of invoking a predicate function of one argument (the name of the next theorem whose pattern has matched the goal) which can "veto" the use of that theorem.

---

10. Note that the differences were ordered (i.e., the sequence of rows in the table of connections was significant), but the operators (the columns) were not. GPS thus had a single, global strategy concerning subproblem selection (which said roughly "do the hardest problems first and don't try a subproblem harder than it's parent problem"). It is easy to imagine extending the system to include strategies which ordered the operators as well. This might be done with either a single, global strategy (ordering the columns), or a number of more specific strategies (ordering the operators within each row).

11. STRIPS also offered a number of other mechanisms for guiding program performance, but they lie outside the focus of this discussion.

```
(THGOAL (ON A B) (THUSE QuickStack CarefulStack))

(THGOAL (ON A B) (THTBF IsItAFastTheorem))
```

Figure 5 -- Two alternate forms of PLANNER advice for the same goal pattern

In PLANNER, advice about refinement is associated with a specific goal (or task) in a specific context. This makes possible different advice for different goals, or even different advice for the same goal in two different contexts. This level of indexing makes it possible to embed numerous specialized, local advice rather than a single, global heuristic as we have seen above.

The depth-first search with backtracking that was built into PLANNER restricts the utility of the filter mechanism slightly, since it is simply a veto based on a single theorem name. That is, final judgment is passed on each potential theorem in turn, hence it is not possible for instance to make comparisons between potential theorems, nor to pass judgment on the whole group and choose the one that looks (by some measure) the best.

For our purposes, CONNIVER's invocation mechanism is similar in many respects, except that a pattern-directed call yields a "possibilities list" containing *all* the KSs that matched the pattern. While there is no explicit mechanism parallel to PLANNER's recommendation list, the possibilities list is accessible as a data structure and can be modified with list manipulation operations to reflect any judgments that can be made concerning the relative utility of the KSs retrieved.

NASL's "choice rules" are the closest to meta-rules in overall design. They offer a means of encoding advice about choosing between alternative KSs applicable to the same task and are associated with specific tasks. As we describe below, NASL's choice mechanism is somewhat more restricted than the evaluation facility of meta-rules, but its "rule together" mechanism (which permits reformulation of a problem) allows a more general facility for subgoal manipulation.

10.2 Dimensions of comparison

The systems reviewed differ in several fundamental ways. For our purposes, four dimensions of comparison are especially relevant: level of indexing, uniformity, accessibility, and degree of explicitness.

Except for procedure invocation (a degenerate case since no choice is available at runtime), all of the techniques mentions use either a single, global evaluation function, generally hardwired into the interpreter (traditional production rules, GPS, STRIPS, QA3, and HEARSAY-II), or they allow a multiplicity of local evaluation functions, generally associated with specific goals (PLANNER, CONNIVER, and QA4). We will see below that level of indexing has an impact on both the power and efficiency of a strategy encoding mechanism.

By uniformity, we mean, Is strategy information represented in the same way as object-level knowledge? For all the languages and systems reviewed above except QA3, the answer is "no". Advice in PLANNER, for instance, is encoded in the "use" and filter constructs, not as standard PLANNER theorems. Some of the languages and systems (CONNIVER, QA4, and NASL) come close, but fail to offer any specific features that suggest or support this idea. Only QA3 (and, as we will see, meta-rules) uses the same language at both the object- and meta-levels. Such uniformity offers a number of advantages (Section 11.1).

In examining how *accessible* the strategies are, we ask Are they encoded as program-level objects (as in the local strategies of PLANNER, CONNIVER, and QA4), or are they embedded in the interpreter (as in the global strategies of production rules and STRIPS)?

Finally, it is useful to examine how *explicit* the strategy encoding is in each formalism. There are a number of possibilities, ranging from encodings that make clear the criteria used in the strategy, to encodings that indicate only the desired end effect, and on to encodings in which

even the desired effect is unclear. In HEARSAY-II, for instance, the focus of attention mechanism is a function of five clearly specified criteria. By contrast, the "use" mechanism of PLANNER makes clear only which theorems are desired in which order -- the motivation for choosing that particular subset and ordering is nowhere represented. Finally, as we will see in more detail in Section 11.2.1, in some multiple priority level agenda designs, execution priority levels can be manipulated to produce desired invocation control effects (such as ordering the KS in a specific sequence), but the encoding is so indirect that even the intended result is often obscure.

11 CONTROLLING INVOCATION

11.1 The utility of this approach to invocation control

Some aspects of the invocation control machinery provided by meta-rules are common to a few of the systems reviewed above. The points in common are: (a) the entire set of plausibly useful knowledge sources (the PKS set) is available for examination and modification, (b) refinement is effected via manipulation of the PKS set, (c) refinement information is associated with the knowledge indexing point in use at the object level, and (d) the representation of all levels of knowledge is the same. These ideas provide several interesting benefits; a brief review of them will show why we chose them as a starting point for the design of meta-rules.

11.1.1 Availability of the PKS set

First, having available the entire PKS set makes possible comparative judgments. That is, using meta-rules (or CONNIVER or QA4), it is possible to write a strategy that indicates that one kind of KS is likely to be more useful than another. If the entire set is not available, comparative judgments can't be made, as for example with PLANNER's theorem filter, which can only accept or reject candidate operators one-by-one.

11.1.2 Refinement as manipulation of the PKS set

Second, invocation control based on the idea of manipulating the PKS set presents an approach to knowledge organization and use that appears to offer significant leverage. Gains in performance can be realized by adding to a system a store of (meta-level) knowledge about which chunk of object level knowledge to invoke next. Considered in terms of heuristic search, we are talking about the difference between "blind" search, and one guided by heuristics. The advantage of even a few good heuristics in cutting down combinatorial explosion is well known.

Consider, too, that part of the definition of intelligence includes appropriate use of information. Even if a store of (object level) information is not large, it is important to be able to use it properly. Meta-rules provide a mechanism for encoding strategies that can offer additional guidance to the system.

11.1.3 Indexing of refinement information

Third, associating strategies with the object-level knowledge indexing point -- in this case goals -- makes it possible to specify numerous heuristics for a given goal without imposing any overhead in the search for any other goal. That is, there may be a number of (potentially complex) heuristics describing the best rules to use for a particular goal, but these will cause no computational overhead except in the search for that goal.

More generally, strategies associated with one indexing point do not impose overhead in the computation triggered by another indexing point. This is equally true whether the indexing points are goals, data, differences, etc.

Associating strategies with the knowledge indexing point also provides an interesting perspective on organizing and using knowledge to guide search. In most cases, knowledge used to guide search is encoded in the search algorithm itself and as a result must be fairly general, in order to be applicable to the entire space. Our perspective suggests that, while general search strategies can be encoded in the search algorithm, more specific strategies can be easily encoded and efficiently used by associating them with nodes in the search space. As a result, the search space has an interesting characteristic. At each node, when the system has to choose a path, there may be available information advising about the best path to take. There may therefore be available a body of knowledge to guide the search, in addition to the knowledge already embedded in the code of the search algorithm. The additional information is organized around the objects which form the nodes in the search space --- in additional to a smart algorithm, we can also have a "smart space".

11.1.4 Uniformity of representation

Like QA3, our framework presents a uniform encoding of knowledge -- the same representation (in this case rules) is used to represent both object-level knowledge about the domain and meta-level knowledge about control.[12] This makes the treatment of all levels of knowledge the same, which in turn offers several advantages.

For example, it simplifies the extension to arbitrary levels of meta-rules, making the changeover straightforward both conceptually and with respect to implementation. For instance, there is a single mechanism that allows rules at one level to conclude about rules at the next lower level. Similarly, there is a single (very simple) code examining technique that is used by all levels of rules to examine rules at lower levels. Uniformity of knowledge encoding thus helps cut down on the amount of machinery necessary.

The power in using rules to guide rules applies at multiple levels as well. There is leverage in encoding heuristics that guide the use of heuristics. Thus, rather than adding more heuristics to improve performance, we might add more information at the next higher level concerning effective use of existing heuristics.

Other benefits arise from uniformity. For example, work on TEIRESIAS included development of capabilities for explanation of program performance, based on displaying the relevant object-level rules. The uniform encoding of knowledge eased the task of extending this to meta-rules, making possible an interesting capability. In addition to being able to display the object-level rules used during a consultation, TEIRESIAS can similarly display the meta-rules, thereby making visible the criteria used by the performance program in "deciding how to do what it did". Knowledge in the strategies has become accessible to the rest of the system, and can be displayed in the same fashion (Figure 6).

Work on TEIRESIAS also explored knowledge acquisition (see [13]), but we have not yet attempted the interactive acquisition of meta-rules. Their slightly more complex syntax requires generalizing some of the routines developed for acquisition of object-level rules. It does not appear to be a difficult problem, however, and beyond it lie some interesting issues. Given the effectiveness of acquisition of object-level rules in the context of a bug (illustrated in [13]), can the same be done with meta-rules? What are the superficial manifestations of a bug that suggest the need for a change in the way established knowledge is used (rather than a change in the content of the knowledge itself)? We have considered this briefly, and the most obvious case seems to be one analogous to human behavior. When someone inexperienced in a domain takes

---

12. Also see [52], which describes a system exploring the use of meta-level knowledge from a different perspective, and which discusses other advantages of uniformity of representation.

a plausible, but (to the expert) inappropriate approach to a problem, the expert can often spot it and inquire why the novice took that approach. Similar situations may be encountered in which the program starts off on a plausible, but incorrect track; the appropriate remedy may be a new meta-rule. This appears to be one interesting direction for future work.

```
10) Amount of investment experience? (slight = less than one year;
    moderate = 2 to 4 years; extensive = more than 4 years)
**MODERATE

11) Does the investor subscribe to the Wall Street Journal, Barron's, or
    any securities market newsletter?
**NO

12) What has the trend of the index of LEI been over the past several months?
    (rising, falling, mixed)
**WHY

[i.e. Why is it important to determine the trend of the LEI index?]

[1.0] This will help determine a strategy for deducing the best stock to
      invest in.
If
 [1.1] the the LEI index has been rising steadily for the past several months,
       and
 [1.2] there are rules which mention in their premise that the economy may
       be heading for a recession,
then
 it is likely (.7) that each of them is not going to be useful.
 [METARULE001]

**HOW [1.0]

[i.e. How is the strategy for deducing the best stock to invest in determined?]

At that point METARULE001 was being used.
   In addition, the following were also used:
   [1.3] METARULE002 indicated it is very likely (.8) that the rules which
         mention high risk should be used after rules which mention low risk.
```

Figure 6 -- Example of meta-rule explanation

11.2 The novelty of our approach and resulting benefits

Our approach to representing and using invocation control information is novel with respect to the use of an explicit, high-level representation, the use of localized representation of control information, and the level of support mechanism provided by the framework.

11.2.1 Explicit representation of invocation control

A central theme of this work is the explicit representation of information about controlling invocation. Meta-rules provide a way of accomplishing such explicit representation. By explicit we mean that the encoding makes apparent both the individual criteria used to decide on a particular pruning or reordering, as well as any interrelationships between the criteria. Consider as a contrast PLANNER's "use" mechanism. As noted earlier, it specifies the order in which potentially applicable theorems should be tried, but does not indicate the rationale behind

that ordering (e.g., are the fastest theorems first, those most likely to succeed, or are there possibly several criteria all contributing to the final order chosen?)

Implicit encodings of this sort often result when the available set of invocation mechanisms does not allow direct expression of all of the behavior the programmer may want. In such cases he may resort to various devious techniques. One popular approach is that of getting a multitude of effects from a single mechanism. For instance, where the retrieval and refinement phases are combined and KSs are retrieved via pre-computed lists, a common approach is to hand-tool the ordering of these lists to achieve effects not otherwise available in the existing formalism. For example, where goal-directed invocation is accomplished by using pre-computed lists of operators, hand-tooling those lists can add a range of other refinement effects. In [50], for instance, the GOALCLASS lists were hand-ordered to insure that the fastest operators were invoked first. The analogous lists in MYCIN have in the past been hand-tooled to effect a number of partial orderings on the rules that are invoked.

A similar example arises when using a multiple priority level agenda of the sort described in KRL [2]. Suppose, for example, we wanted to insure a particular partial ordering of processes to be put on the agenda. Note that there is no way to say explicitly, *make sure that these processes (in set A) are executed before those (in set B).* Instead, we have to be indirect, and could for instance assign a priority of 6 to the rules in set A, and a priority of 5 to those in B.

There are a number of problems associated with trying to do these sorts of indirect encodings of refinement, all of which seem to arise from the fact that information is unavoidably lost by the indirection involved. Note that in all the cases above, the intent of the hand-tooling and indirect priority setting is nowhere recorded. The resulting system is both opaque, and prone to bugs. For instance, in the agenda example, after the priorities have been set, it will not be apparent *why* the processes in A were given higher priority than those in B. Were they more likely to be useful, or is it desirable that those in A precede those in B no matter how useful each may be?

A class of bugs (which we call "partial order" bugs) can arise in this setting due to both execution-time events and events in the long-term development of the program. Consider for instance what happens if, during a run, before we get a chance to invoke any of the processes in A, an event occurs which makes it clear that their priority ought to be reduced (for reasons unrelated to the desired partial ordering). If we adjust only the priority of those in A, an execution-time bug arises, since the desired relative ordering may be lost. Yet there is no record of the necessary connection of priorities to remind us to revise those for set B also. A similar problem can arise during the long-term development of the program, since, after a while, even the programmer who set the original priorities may forget what motivated them. If he then attempts to introduce another partial ordering of the KSs by juggling priorities, he may end up modifying the priorities of the KSs in set A without the necessary adjustments to those in B.

The problem results from (a) reducing a number of different, incommensurate factors to a single result and (b) failing to keep a record of how the result was obtained. This is true both for the agenda and the GOALCLASS mechanisms. In the agenda case such disparate concepts as individual utility and relative ordering are reduced to a single, absolute priority level; in the GOALCLASS case they are reduced to the position of an operator in the (totally ordered) GOALCLASS list. In neither case is a record of the computation maintained.

The loss of information causing the problem often results where retrieval and refinement are accomplished by a single mechanism. In the GOALCLASS and MYCIN examples, for instance, the list used for retrieval is also pressed into service to represent refinement information. But a simple list is not adequate for this task. It is, for example, totally ordered (and we might want to represent partial orders), it offers no mechanism for keeping track of ordering criteria, etc. As a consequence, it ends up encoding only the final result of refinement and hence loses

information.

By separating out refinement and retrieval, we are more likely to supply a separate and sufficiently powerful representation for refinement information, reducing the temptation to make do with whatever mechanism happens to be left around from the retrieval phase. Meta-rules offer one mechanism for representing refinement information and for maintaining a record of why a set of KSs has been queued in a particular order.

Two other research efforts have dealt in varying levels of detail with this concept of explicit representation of control information. The GOLUX [21] project has emphasized both the utility of separating logic and control and the advantages arising from having a language for expressing control flow information. Several of the control primitives suggested there (e.g., pruning or ordering operators) turn out to be quite similar to those used in TEIRESIAS. It appears, however, that the language never reached a stage of effective implementation.

The work on AMORD [14] has also concerned itself with the issue of explicit representation of control information. This arose because, to the designers of AMORD, PLANNER's invocation appeared too implicit. PLANNER keeps goal-tree information represented implicitly in the interpreter and uses a built-in depth-first search with chronological backtracking. These constructs can produce significant amounts of unnecessary search.

In response, AMORD represents goals explicitly in the database and reverts to a more basic control mechanism that has virtually no built-in preconceptions. Even such a basic notion as sequencing of goals is no longer taken for granted.[13]

This view of control has two interesting consequences. First, it becomes possible to use a wider range of control regimes then is available in PLANNER (e.g., removing sequencing as a built-in concept makes it possible to write code which works on two goals alternately).

But second, by reverting to a more primitive view of control, AMORD requires the user to write all his own control constructs, and this is especially difficult in a world where not even sequencing is taken for granted. As a result, while AMORD *lets* the user control invocation in situations where PLANNER did not, it does not *help* the user with this formidable task. A critical element it has not yet supplied is some vocabulary of control concepts, a collection of primitives that the user can employ to represent control. Such a vocabulary becomes especially important in a world where even the basic constructs have to be rebuilt from scratch and then invoked explicitly. Without it, the promise of explicit control and explicit representation of control is somewhat hollow: they have made it possible, but they have not made it easy.[14]

11.2.2 Localized representation of invocation control

Meta-rules also offer the advantage of *localizing* all of the refinement information. Note that juggling priorities (as in the agenda example above) means trying to achieve a global effect via a number of scattered local adjustments. This is often quite difficult, and can be very hard to change or update. Localizing each such refinement criterion in a single meta-rule makes subsequent modifications easier, since all of the information is in one place -- changing a criterion can be accomplished by editing the relevant meta-rule, rather than searching through a program for all the places priorities have been set to effect that criterion.

---

13. The decision to work on a goal is independent of the announcement of the goal, hence there is no built-in mechanism which commits the system to work on goals in the order in which they are announced.

14. The work on meta-rules is not entirely immune from this criticism either, even though it does not put the user in a world where basic control constructs have to be rebuilt. The meta-rule framework offers a very basic vocabulary for control strategies, in the notions of pruning and refinement of the PKS, but this is clearly only a beginning. We comment further on the limitations of the current vocabulary in Section 11.3.

11.2.3 Support mechanisms

Our perspective on invocation control offers the user a conceptual framework and the beginnings of a vocabulary for expressing refinement information. This makes it easier to formulate and encode the appropriate knowledge.

For instance, inference rules of the sort illustrated, while not necessarily optimal in all respects, do provide a useful medium of expression. In the form we have used them they provide a high level language, and like all high level languages make possible direct statements phrased in domain specific terms.

The rule-based format is also effective where decisions can be made by combining the contributions of a number of independent criteria. In that case it provides a convenient mechanism for expressing and using those criteria.

The use of the model of confirmation described in [43] makes it possible to write rules which make different conclusions about the best strategy to use, and then have the underlying model of confirmation to weigh the evidence. That is, the strategies can "argue" about the best rule to use next, and the strategy that presents the best case (as judged by the confirmation model) will prevail.[15]

The judgmental character also allows the novel possibility of both inexact and conflicting statements concerning relative order. We might, for instance, have two meta-rules which offer different opinions about the order of two sorts of object level rules, indicating that there is evidence that "`subset X should probably (.6) be done before subset Y`", and that "`subset Y should probably (.4) be done before subset X`". Once again, the underlying model of confirmation will weigh the evidence and produce an answer.

There is some support available too from the general view of strategies as meta-level knowledge. Viewing invocation control as knowledge about knowledge encourages a broader perspective on the issue than might otherwise be taken. That is, the more empirical "how do I get the right thing to happen next" programming perspective encourages local patching. Thinking about strategies as information about object-level KSs offers a view that encourages stating more general principles.

Finally, there is some benefit gained from the (so far very rudimentary) meta-rule vocabulary (e.g., concepts like KS *utility*, *ordering*, etc.). It provides a set of predefined concepts with which to express strategies and can, as a result, ease the task of stating explicitly strategies which have previously been only implicit. Any substantive impact along these lines will of course require a much more fully developed vocabulary; the limitations of the present set are discussed in below.

All of these mechanisms have provided useful support and offered a foundation for expressing strategies of the sort shown in Section 7. As a concrete illustration of their utility, consider viewing traditional resolution theorem proving strategies in these terms.

In that domain the set of plausibly useful knowledge sources consists of: all of the clauses in the original axioms, clauses resulting from the negation of the theorem, and any derived via previous resolutions. There is an obvious saturation problem here because there is in the basic resolution method no selective retrieval (indeed, much work has been devoted to developing ways of being more selective; see, e.g., [6]). Many of the strategies developed can be viewed in terms of pruning and reordering the PKS set.

Consider for example set-of-support and unit preference. Unit preference can be phrased as

---

15. The current model of confirmation may be too simple to provide any extensive ability to "argue" (see, e.g., [1]). In particular, it uses a uniform numeric encoding scheme and is not suited to situations where the rationale behind an argument is as important a factor as its strength. In that case some of the techniques of the dependency-based methods (e.g., [15]) may be useful.

```
if a clause consists of a single literal,
then it should definitely be used first.
```

Set of support becomes

```
for a pair of clauses, A and B,
if A is not a member of the support set, and
   B is not a member of the support set,
then it is definite that A and B should not be resolved together.
```

These statements are reasonably clear and comprehensible, in part because of the simplicity offered by the high level language approach, and because of the utility of the notions of pruning and reordering. The advantage of the high level language approach becomes more evident when compared with alternative formulations like the predicate calculus approach used in [20]. There, the expression of unit preference requires a rather opaque theorem eleven lines long. The difference in length and complexity arises because the predicate calculus formulation requires non-trivial mechanisms to describe the idea of ordering and to deal with issues like the commutativity of resolution.[16]

Our approach includes as a primitive the notion of ordering the KSs (clauses) and can avoid details like commutativity by choosing the appropriate data structures for use in the code that carries out the resolution. While all of the information in the predicate calculus version must be represented somewhere in our approach, the point is that much of that information can justifiably be regarded as detail and can be taken care of at a lower level. This yields a more comprehensive statement of the strategy.

In addition to simplicity, this form of encoding and using strategies is useful because of its "additivity". That is, we can imagine a collection of several such pruning and reordering rules, all stated individually and all offering advice about pruning and reordering the set of clauses. Such a collection is additive in the sense that with, say, the unit preference rule alone, we get the unit preference strategy. Add the set-of-support rule and we obtain the merging of these two strategies; other collections of strategies can similarly be explored.[17]

11.3 Control: limitations of the current implementation

The current system has a number of limitations and shortcomings. Problems arise (a) from the view of strategies as operators modifying the current set of plausibly useful KSs (the current PKS set), (b) from viewing this modification in terms of simply pruning and reordering the set, (c) from the lack of an extensive vocabulary for describing knowledge, (d) from the failure to distinguish between domain independent and domain specific meta-level knowledge, and (e) from the lack of a mechanism for controlling subgoal selection. Some of these are artifacts of the current implementations; others result from difficult fundamental problems.

---

16. The strategy in [20] is also a slightly more general form than standard unit preference.

17. This design as it stands is not particularly efficient for theorem proving and hence would not be a practical tool in that domain. One possible way of dealing with this would be a "strategy compiler" which transformed the statements shown into more efficient code for selecting clauses to resolve.

Also, given the long time typically required for development and completeness proofs for such strategies, the opportunity for empirical exploration of additivity offered by our design is less important in this domain. Resolution thus offers a useful example of the utility of a high level representation (the strategy becomes more comprehensible), but illustrates a domain in which changes to the set of strategies are infrequent enough that little use is made of additivity.

For domains in which development of new strategies was much more common, the additive character of our strategy encoding scheme might prove quite useful.

11.3.1 Strategies viewed as operations on the curent PKS set

First, if strategies can only operate on the current PKS set then it is difficult in the current system to implement strategies to control a "line of thought" (i.e., a sequence of KS invocations stretching over several cycles of the system). Consider, for example, the notion of "unit preference with a level bound" as a resolution strategy. It requires that we keep track of how many times the unit preference strategy has been used, and then be prepared to take alternative action if it reaches its limit without success. While it does not appear difficult to add such a capability to our system, the current implementation does not have it.

11.3.2 Control as pruning and reordering

Second, we might ask whether simply pruning and reordering the PKS set is a powerful enough mechanism. Might we not also need the ability to set up more complex plans, as for example, conditional statements rather than simply a fixed ordering of the KSs? It may also prove useful to express resource allocation information in meta-rules: In addition to having meta-rules which concluded about the likely utility of an object-level rule, meta-rules might also indicate that "if this rule is going to succeed, it will do so within N cycles". This would have the advantage of making the information more explicit than it has typically been in (see, e.g., KRL [2] and AM [28][18]). While we have not as yet found any of this necessary, it is likely to be required in the long run on more complex problems than we have yet attempted.

11.3.3 The need for a vocabulary of control concepts

A third and very important limitation of the current system arises from the rudimentary state of our language for expressing meta-level knowledge. We have thus far only a few conceptual primitives for describing control (e.g., *order* and *utility*) and a perspective on the issue (viewing strategies as meta-level knowledge) which may help indicate how to find additional primitives.

Viewing strategies as meta-level knowledge aids by emphasizing that the task is to describe characteristics of knowledge (rather than characteristics of the domain, as is the case for object-level knowledge). Some of those meta-level primitives might deal with general properties of a KS like its preconditions, main effect and any side effects. Other relevant properties might be suggested by the structure of the object-level primitives. For instance, as noted earlier, the performance program uses both consequent and antecedent rules; each rule is composed of a premise and an action; these, in turn, are made up of clauses; and the clauses are built from predicate functions, attributes, objects and values. Each of these suggests several possible meta-level primitives: Is the rule an antecedent or consequent rule? How many premise clauses are there? Which functions, attributes, etc., does the rule employ? In the same way, the components of any sort of KS could provide hints about potentially useful primitives for characterizing it.

There is also the interesting question of what is to be the primitive element of computation which is to be controlled. In our framework, as we have seen, the basic control cycle is a rule invocation, and control of invocation consists of modifying the PKS set. In the work on GOLUX, based as it was on predicate calculus as a representation, the basic element of computation was a proof tree, and statements about control were statements about the

---

18. AM even used the same number as both the priority of a task and its resource allocation, yet they are clearly different concepts. The task given the highest priority might happen to be one which can be accomplished either very quickly or not at all. For efficiency's sake, then, it should receive a relatively small resource allocation.

characteristics and form of those proof trees. It may be that each kind of inference engine will have its own basic computation cycle and hence its own perspective on the primitive element of computation to be controlled.

11.3.4 Domain independent and domain specific meta-level knowledge

There is something mildly unsettling about a framework which fails to distinguish between domain independent and domain specific meta-level knowledge. "Try rules with CF = 1.0 first" seems sufficiently different from "Try rules which mention high risk first" that some distinction should be made. This would be at least a useful knowledge engineering technique, since the "CF = 1.0" rule is presumably valid across several domains and hence could remain a part of a "core" system which might be instantiated in a number of domains.

11.3.5 Subgoal selection

Another limitation arises from the lack of a mechanism for doing subgoal (subproblem) selection. In MYCIN subgoal ordering is hardwired in by the choice of a depth-first search and the ordering of preconditions in the premise of a rule. In the same sense that standard procedure invocation offers only an indication of which KS should be chosen in which order, with no record of why, so MYCIN has no record of why the preconditions of a rule are ordered as they are (and TEIRESIAS inherited this limitation from working in that context).

This is clearly undesirable. Earlier we saw the utility in making explicit the decision about which KS to apply next. It would prove equally as useful to make explicit the decision about which subgoal to work on next. A base of rules about subgoal selection should be developed and a place provided for them in the revised system design. We might even adopt the interesting perspective in the NASL system, which has choice rules for reformulating tasks, and which can replace existing goals with a single new goal intended to be a synthesis of the old ones.

11.3.6 Character of the domain

A final, somewhat obvious limitation is that the application domain must support the use of multiple levels of rules. This requires a domain with a certain level of complexity. There must be enough object-level knowledge sources that meta-rules are worth the computational overhead, for example, and there must be some basis for pruning or reordering the object-level KSs that leads to more effective problem solving performance. This support also presumes that strategy knowledge about the domain can be formalized and stated in terms that allow its use in guiding program performance. Thus the domain must not be so simple that exhaustive invocation is computationally feasible, nor so informal that it is difficult to determine even the basic meta-level primitives.

12 IMPLEMENTING INVOCATION CONTROL

The previous section considered the utility of refinement as a means of controlling invocation. It examined the benefits that resulted from our framework for organizing and using knowledge about refinement. We consider here a somewhat more detailed issue, focussing on problems that arise in implementing meta-rules. We consider in particular the implications that follow from meta-rules' ability to select object-level rules by direct examination of object-level rule code. We begin by examining the difference between various approaches to referring to

knowledge sources.

12.1 Reference by name vs. reference by description

Since strategies (in any form) are used to direct the invocation of object-level KSs, they must have some way of referring to them. Two fundamentally different approaches have typically been employed; we term them *reference by name* and *reference by description.* The former lists all KSs explicitly, while the latter offers a predicate indicating required characteristics.

Meta-rules effect reference by description. As illustrated, they make conclusions about a class of rules that is specified by describing relevant characteristics. `METARULE002`, for instance, refers to `"rules which mention high-risk in their premise"`. PLANNER's theorem base filter construct is similar in this respect, since as we have seen, it allows the specification of an arbitrary predicate to filter the selection of theorems to be applied to a goal. As noted earlier, the "use" (`THUSE`) construct of PLANNER uses reference by name, allowing the programmer to name one or more theorems that are likely to be especially useful.

There are numerous advantages to reference by description, primarily in terms of the ability of the knowledge base to accommodate changes. These are explored later in Section 12.4. Here we examine a more detailed point: the implications involved in two different ways of accomplishing reference by description.

12.2 External descriptors vs. content reference

One way to accomplish reference by description is via a set of external descriptors. A number of different characteristics could be chosen and each KS described by the programmer in terms of them. For an ordinary procedure, for instance, the descriptor set might include elements describing the procedure's main effect, any side effects, preconditions for its use, etc. Strategies would then describe the relevant class of procedures in these terms.[19]

The second implementation is by direct examination of KS content and is illustrated by meta-rules. For instance, when `METARULE002` refers to `"rules that mention high-risk"`, the relevant set is determined by retrieving the body of each of the rules in question and examining it directly to see if it contains the desired item.

This general notion of allowing one part of the system to examine the rules (code) executed by other parts is based on three main ideas. First, there is the concept of the unity of code and data structure, first suggested in the notion of a stored program computer and later made convenient by LISP. Second, the rules must be stored in a comprehensible form. In this case that means interpreted LISP code, written in a stylized form; but in general any relatively high-level language will do. Finally, the syntax and some of the semantics of that high-level language must be represented within the system, to be used as a guide in examining the code. In the current example, the syntax is represented by both the template associated with each predicate function (which allows each function to describe its own calls) and the stylized form of rules. Semantics are supplied in part by internal data structures which indicate, for example, that `RISK-LEVEL` is an attribute.

---

19. See [25] for some speculations on additional uses of external descriptors.

12.3 Content reference: example

To see how content reference is implemented, we consider once again the meta-rules in Figure 3 and examine in more detail how they work.

Recall that we were trying to determine which stock to invest in, and assume that the list (L) of relevant object-level rules starts with

```
(RULE076 RULE042 ...)
```

Figure 7 - The list of relevant object-level rules

Rules 76 and 42 are shown below.

```
RULE076
If    [1] the risk level of the investment should be high-risk, and
      [2] the time-scale of the investment is short-term, and
      [3] the investment area is gambling stocks,
then Bally is a likely (.6) choice for the investment.

($AND  (SAME OBJCT RISK-LEVEL HIGH-RISK)
       (SAME OBJCT TIMESCALE SHORT)
       (SAME OBJCT INV-AREA GAMBLING))
(CONCLUDE OBJCT STOCK-NAME BALLY .6)


RULE042
If    [1] the investment area is the food service industry, and
      [2] the desired size of the company is small, and
      [3] the timescale of the investment is short-term,
then Wendy's International is a likely (.5) choice for the investment.

($AND  (SAME OBJCT INV-AREA FOOD-SERVICE)
       (SAME OBJCT SIZE SMALL)
       (SAME OBJCT TIMESCALE SHORT))
(CONCLUDE OBJCT STOCK-NAME WENDY .5)
```

Figure 8 -- Rules 76 and 42

Before invoking the list of object level rules, we would find that there are two meta-rules (those in Figure 3) associated with this goal and we attempt to invoke them. Assume that the LEI index has not been rising, and that we are dealing with someone who is more than 60 years old. The first clause of METARULE001 fails and hence we do nothing more with that rule.

The first clause of METARULE002 succeeds and we continue by evaluating the second clause:

```
(THEREARE RULES (MENTIONS OBJCT EITHER HIGH-RISK) SET1)
```

This proceeds by evaluating the predicate clause (the MENTIONS part) on each of the rules in L and collecting (in SET1) those for which the predicate is true.

The interesting issue here is, How do we know if the predicate is true? There are two ways in which it can be satisfied. We may discover that the term HIGH-RISK is mentioned explicitly, or we may infer that it is mentioned.

As an example of the first way, consider evaluating the MENTIONS predicate on rule 76. This operates as explained in Section 8.1. The template for each of the clauses of rule 76 would be used to examine the clauses and see if the term HIGH-RISK appears explicitly. This succeeds in the first clause.

The second way of satisfying the predicate -- inferring the presence of a concept -- is illustrated by the evaluation of METARULE002 on Rule 42, and shows that content reference may involve inference processes more complex than simple pattern matching. Here the search for explicit mention of HIGH-RISK fails, and the question becomes, Is the rule talking about a high-risk investment even if it isn't using that term explicitly? To determine the answer we use the standard retrieval mechanism and retrieve all the rules that can tell us about the risk level of an investment. Among them would be the following:

```
RULE101
If   an investment area is sensitive to inflation,
then the risk level of the investment area is likely (.4) to be high-risk.

RULE102
If   an investment area is dependent on raw materials whose prices
     fluctuate widely,
then the risk level of the investment area is very likely (.6) to be
high-risk.

RULE103
If   a stock's dividend rate is stable,
then the risk level of the stock is not likely (.4) to be high-risk.
```

Figure 9 -- Rules about level of risk

Note that these are items of object-level knowledge from the standard knowledge base and might be used in the normal fashion during an investment advisory session. Here, however, they are being used in the context of reasoning about rule 42 (i.e., the object to which they will refer is rule 42). Invoked in this context, a better translation would be:

```
RULE 101
If   a rule mentions an investment area that is sensitive to inflation,
then the rule mentions an investment area whose risk level is likely (.4) to
     be high-risk.
```

and similarly for the others.

The standard invocation mechanism is used and we proceed by back-chaining the rules (see Figure 10). To determine the truth of the premise of rule 101 we use:

```
RULE201
If   an investment area is dependent on numerous minimum wage employees,
then it is very likely (.6) to be sensitive to inflation.
```

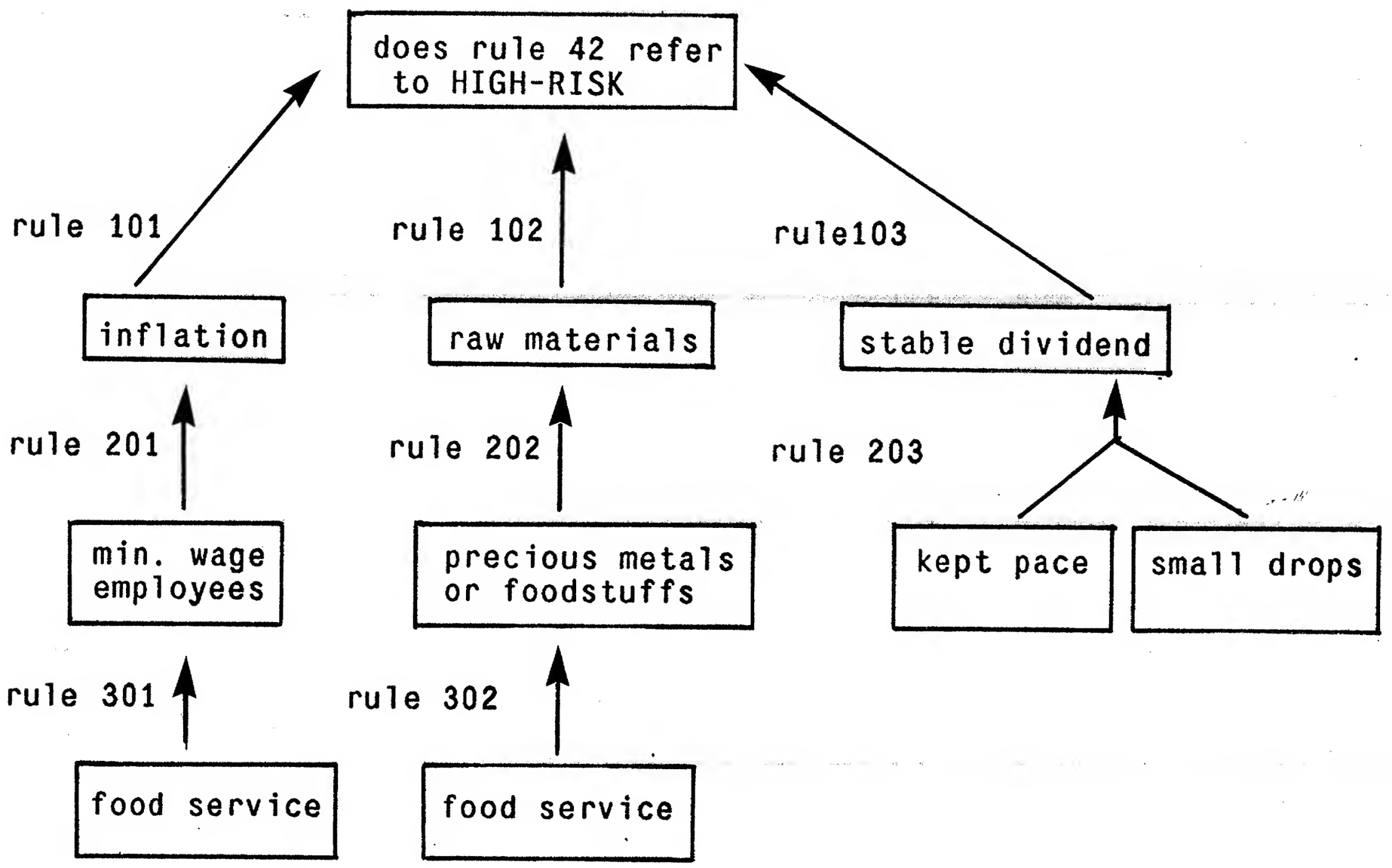
(Minimum wage levels are raised periodically to keep pace with inflation, which means automatic

raises and hence sharply increased costs for industries with numerous minimum wage employees). The truth of the premise of 201 is finally given by rule 301:

```
RULE301
If   the investment area is the food service industry,
then it is definitely (1.0) dependent on numerous minimum wage employees.
```

As shown in Figure 10, Rule 102 leads to Rules 202 and 302, while Rule 103 leads to Rule 203, the truth of whose premise is in turn determined by reference to a data base of stock statistics.

As a result of this reasoning, we discover that Rule 42 meets the condition specified in the premise of METARULE002, and should be collected in SET1 along with other rules which mention high risk either explicitly or implicitly.



RULE202
If an investment area depends on precious metals or foodstuffs as raw materials,
then it definitely (1.0) depends on raw materials whose prices fluctuate widely.

RULE302
If the investment area is the food service industry,
then it definitely (1.0) depends on foodstuffs as raw material.

RULE 203
If [1] the stock's dividend has kept pace with inflation over the past ten years, and
[2] the stock's dividend has never dropped by more than 10% in any one year,
then there is good evidence (.8) that the dividend is stable.

Figure 10 - The reasoning used to infer that rule 42 mentions HIGH-RISK

12.3.1 Lessons from the example

One central point here is that even though Rule 42 doesn't explicitly mention the term HIGH-RISK, we have inferred (using the rules in Figure 9) that 42 does in fact deal with a high risk investment. Note that this was done not by pattern-matching, but by *reasoning about the content of the rule.* That reasoning involved rules that were judgmental (inexact) rather than simply definitional and required combining conflicting evidence (stable dividends pointed to low risk, while all the rest pointed to high risk).[20]

The second main point is that this reasoning was accomplished using the same inference engine that was employed at the object level. That is, the same mechanism is used both to reason (at the object level) about investment selection and reason (at the meta-level) about rule content. This offers a useful economy of machinery.

Third, the mechanism described above can be viewed in terms of a well-known idea used in many systems: When seeking a particular token (or expression or goal) first look in the data base to see if you have it explicitly there already, and if that fails, try inferring its existence via a body of rules (or operators). PLANNER and all the PLANNER-like languages use this, as well as numerous application programs. Note however that, in keeping with our theme of meta-level knowledge, we have moved this process "up" to the meta level. The "data base" we examine is the body of the object-level rule and if the search there fails, we invoke a number of rules to reason about that "data base."

Finally, recall that the standard retrieval and invocation mechanism is employed to effect this reasoning about object-level rules. From this arises yet another benefit of uniformity of representation and uniformity of operational machinery: *multiple uses of knowledge.* Since RISK-LEVEL is an attribute in the object-level knowledge base, it has associated with it a body of object-level rules (some of which were shown in Figure 9), which infer risk level of an investment. During an ordinary advisory session, this knowledge is used in the standard fashion to reason about investments.

But note that this same body of knowledge is applied when the system reasons about whether a rule mentions a particular risk level. That is (Figure 11), the knowledge already in the system for reasoning about things in the domain (potential investments), can as well be used to reason about things in the knowledge base (other object-level rules).

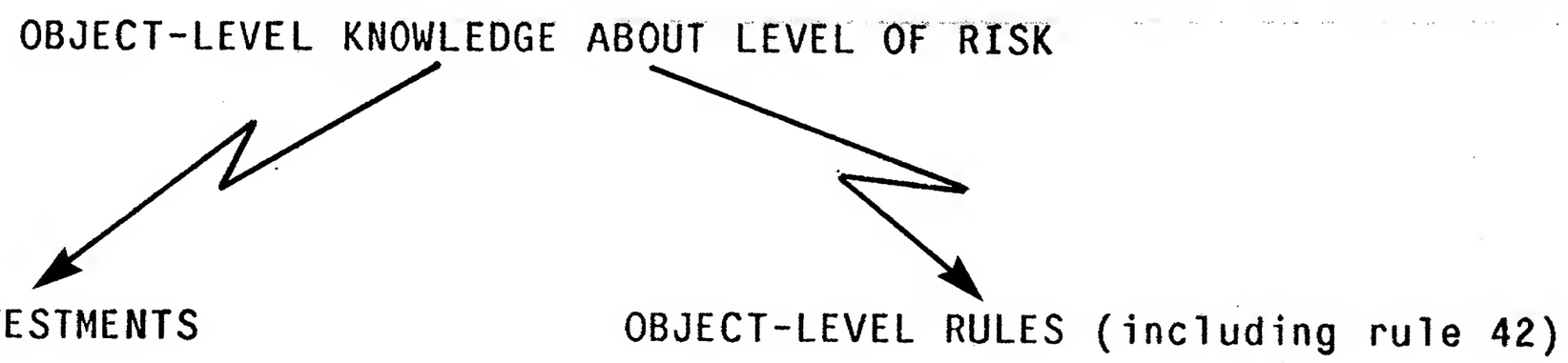


Figure 11 -- Applying knowledge about level of risk in two ways

To complicate the issue, note that that this is true of both the object-level rules and meta-rules. That is, we are trying to determine if RULE042 mentions HIGH-RISK, and call on a set of object level rules (Figure 12) answer the question. But there may also be some

---

20. Note that we need not do these computations at execution time. Assuming that the set of object-level rules is constant over the course of a single run, we can, in between runs, allow the meta-rules to determine the set of object-level rules to which they refer, and replace the descriptions in the meta-rules with those explicit sets. See Section 12.5 for additional comments on efficient implementation.

meta-rules that suggest how to reason about RISK-LEVEL, which are used (in the ordinary fashion) to guide the invocation of the object-level rules. Thus, the same body of (meta-level) knowledge is (a) used to guide the reasoning about investments, and (b) used to guide the reasoning about the knowledge base.[21]

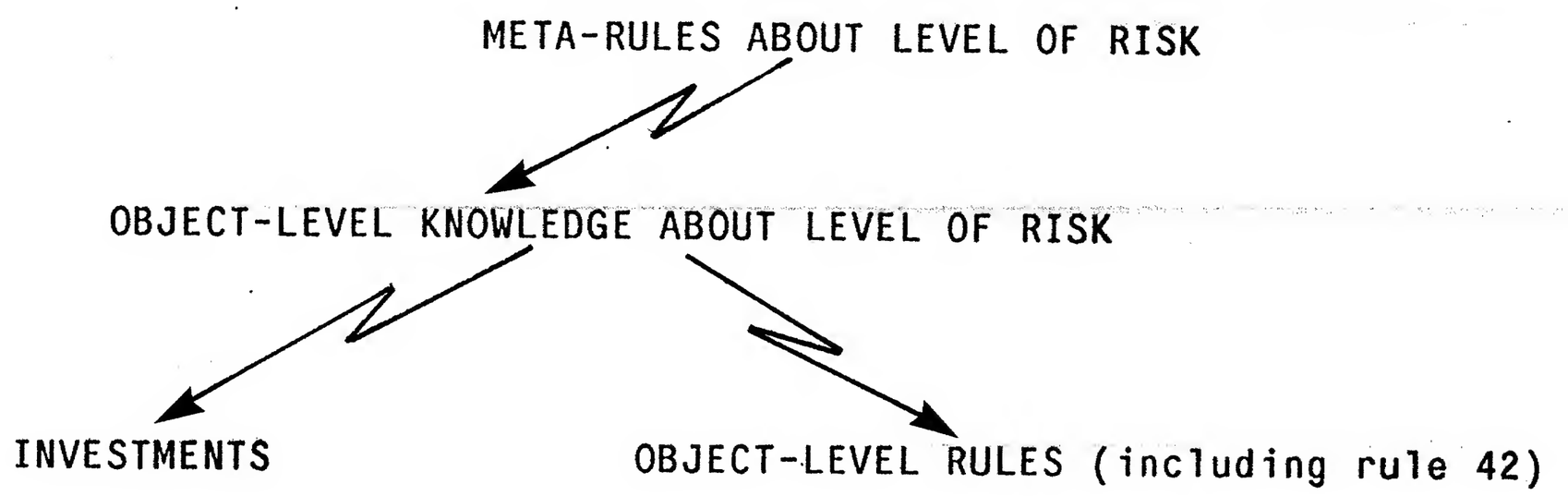


Figure 12 -- Applying meta-level knowledge in two ways

12.4 Benefits of using content reference

There are a number of implications that follow from the use of content reference to express strategy information. The technique makes possible a system with a high degree of *flexibility* in responding to changes, a system in which invocation control has a greater degree of *reliability*, and a system in which the user is freed from the constraint of using only predefined KS reference techniques. We consider each of these in turn, then review the limitations of our approach and the difficulties that remain before the benefits noted can be fully realized.

12.4.1 Flexibility

The choice of a KS reference technique (name, external descriptor, or content reference) can have a significant impact on the difficulty of making changes to a program. This consideration becomes particularly important for application programs with large, constantly evolving knowledge bases, since, as a program gets larger it becomes increasingly difficult to cope with the effects of changes. As we will see, content reference offers a number of advantages in this respect.

We will be dealing here with what we call *compile time flexibility*, i.e., the ability to make changes in the knowledge base between performance runs and then have those changes integrated throughout the system. (Other forms of flexibility are discussed in [8].)

To judge the impact of selecting one or another of the reference techniques, we examine two types of changes: editing or adding an object-level KS to the system, and editing or adding a strategy (meta-rule). We assume in what follows that any modified or new KSs use the existing representation, i.e., no new kinds of KSs are being added. Also, while the impact is

---

21. This use of the same knowledge at different levels has one small drawback in our current implementation. In a somewhat obscure but plausible case, it can lead to unbounded recursive behavior in which the system keeps trying to reason about its rules as successively higher levels. This is easily detected, however, and can be cut off at the appropriate point to allow the system to function properly.

illustrated in the context of meta-rules and object-level rules, the point is more generally applicable to any kind of KS which refers to any other KS, regardless of the level of each. See Table 1 for an overview.

Consider, for example, the effect of editing (or adding) a KS and imagine that strategies used the reference-by-name approach. First, after editing a KS in such a system, all strategies that mention it must be retrieved and examined to see if they still apply, and then must be edited accordingly. Next, since it is also possible that the revised KS should now be mentioned in other strategies, the rest of the strategies must also be examined. Using the external-descriptor approach, we need only update the appropriate descriptors, which would be stored with the KS. In addition, the updating required should be evident from the sort of editing done on the KS itself. All relevant strategies will then reflect these changes. With content reference there is no additional effort of even updating descriptors since the strategies will adjust to the changes found in the edited KS.

Adding a new strategy to the system (or revising an old one) also causes problems for the reference-by-name approach. It is necessary to review all the KSs to determine which ones the new or revised strategy should mention.[22] Using external descriptors, it is possible that no additional effort is required, if the description in the new strategy uses the available "vocabulary" of descriptors. If, however, it requires a descriptor not yet in that vocabulary, we have the formidable task of reviewing all existing KSs and adding to each the appropriate entry for the new descriptor.

Thus there is a fundamental shortcoming in the external descriptor approach because it uses a fixed number of descriptive terms. Since adding a new term to this set can involve a lot of work, it becomes a task that should not be undertaken very often. Avoiding this updating problem is one important advantage of content reference. It gives meta-rules the ability to "go in and look" for any characteristic deemed significant. As a result, the addition of a new strategy with a new meta-level conceptual primitive is a transparent operation.

To illustrate this, consider the following simple example. The performance program's backward chaining of rules produces a depth-first search of an and/or goal tree, where each clause of a rule may possibly sprout a new sub-tree. It might be useful, then, to try first those rules that have a small number of premise clauses. This heuristic could be implemented as a strategy which said something like *try those rules with three or fewer premise clauses first*, and might need a new meta-level primitive to express this. In the external descriptor case, a property indicating the relevant information would have to be added to every rule.[23] Content reference makes the task much easier. We can write a new function that counts the number of premise clauses found in an object-level rule and use the function in a meta-rule. Nothing at all need be done to any object-level rule.

We claim, in addition, that the set of useful meta-level primitives is potentially large and therefore difficult to define a priori. Thus, over the course of most program development periods, it is effectively an open set, to which new members are continually being added. It is thus important to make this task as easy as possible. Where the external descriptor approach requires analyzing each new KS with reference to each of the descriptors, content reference requires simply that the new KS be "added to the pot." It will subsequently be referenced by

---

22. There is a plausible objection to this. It may be claimed that a new strategy is often written with some specific circumstance and purpose in mind and that this clearly restricts the number of KSs that need be considered to a small subset of the total. This is entirely correct. And it is on just such grounds that we would start to build the set of descriptors to be used in the reference by description approach. In part, then, the technique is simply a step toward greater formalization of knowledge already used in informal and ad hoc ways.

23. This may not be a very difficult task if it is possible to write a bit of code that computes the relevant information and then apply it to every rule. In a sense this is what we are doing with meta-rules; see Section 12.6 for further discussion of this issue.

any strategy that describes it appropriately.

To summarize, consider the difference between the first and third rows of Table 1. Note in particular how much easier it is to accomplish a number of standard knowledge base modifications. This can offer a substantive advantage when the knowledge base becomes appreciably large.

TABLE I -- Flexibility Benchmarks

REFERENCE TECHNIQUE	Edit or add object-level KS	Edit or add strategy
Reference by name	Check all strategies to see which should name it	Check all KSs to see which it should name
Reference by description via external descriptors	Update its descriptors	Possibly no additional effort, possibly have to add new descriptor
Reference by description via content reference	No additional effort	No additional effort

12.4.2 Reliability

The choice of a KS reference technique also has a significant impact on a related issue we term "reliability", a measure of the technique's ability to reflect accurately the content of a KS. A rough measure can be obtained by asking: If a KS is referred to because it supposedly has some property (e.g., it supposedly achieves a particular goal), then what happens if I change the KS in such a way as to remove that property? Is the pattern of KS references still unchanged, or does it reflect the modification made to the KS? More generally, does the pattern of references reliably reflect the current KS contents?

Reference by name (e.g., standard procedure calling), for instance, has minimal reliability. There is no formal connection between the name and procedure body, hence arbitrary changes can be made to the body without effecting the pattern of references to the procedure.

Reference by description using external descriptors is similar, since external descriptors have no formal connection to the KS content. This is true of the varieties of pattern-directed reference used in languages like PLANNER and QA4, which, although they avoid KS names and instead use descriptors like goals, data patterns, events, etc., are still no more reliable:

> The subroutines are not referenced by their names. Instead they are called because *they accept arguments with a certain structure*, and because *the programmer claimed* that they will solve goals of a certain class.
>
> from the QA4 manual [40], (emphasis added)

Content reference offers a higher degree of reliability because of its direct reference to KS code. A meta-rule that refers to `"rules that mention high risk in their premise"` will retrieve an object-level rule not because of the rule name, the rule number, or other external descriptor, but because examination of the code of that object-level rule reveals that the rule does in fact mention high risk.

This sort of reliability in the KS reference mechanism can help avoid at least one source of bugs that often occurs in programs as they are developed. When a KS is modified, the modification may make the KS inapplicable to some of the tasks to which it was previously applied. With content reference, the effects of modifications can automatically be reflected in subsequent references to the KS.[24]

12.4.3 Freedom from predefined reference criteria

Finally, content reference offers an added degree of freedom that is useful for defining new reference criteria.

Most previous languages and systems have hardwired in one or another KS reference technique. Standard procedure calling, for instance, is strictly by name and offers no mechanism for other reference techniques. Standard pattern-directed reference is similar. PLANNER, for instance, has three categories of patterns (antecedent [goals], consequents [assertion events], and erasing [erasing events]); all references have to be one of these types.

CONNIVER was the first exception to this. In addition to its standard collection of pattern types (goals, assertion and erasing, renamed as if-needed, if-added, and if-removed), it also has the `METHOD-TYPE` construct. This allows the user to define new categories of patterns which represent new criteria for referring to a KS. If, for instance, it were useful to refer to a

---

24. Note that reliability is distinct from flexibility: reference by external descriptors, for instance, is more flexible than reference by name, but is no more reliable.

KS by some of its side effects (e.g., which global variables it changed), the METHOD-TYPE construct could be used to set up the machinery to allow side-effect-directed retrieval.

Note, however, that patterns are still external descriptors and hence lack any formal relationship to the actual KS contents. A KS may have associated with it a pattern indicating, for example, that it modifies several global variables, quite independent of whether it does in fact modify them.

Content reference has an analogous ability to define new reference criteria, but because it is implemented by examination of the KS code itself, it is more "reliable" in the sense defined earlier.

Thus where reference by name in effect points to a specific procedure, reference by description allows a much more general mechanism. It allows us to say "give me any KS that fits the following description." By writing the appropriate sort of description, we can have reference to goals, side-effects, etc. -- in short, reference to any one of or a combination of factors.

The criteria for KS reference are therefore no longer predefined and embedded in the language interpreter, but can be specified and changed (even dynamically) by the user himself. Consider, for example, the second clause of the premise of the meta-rule in Figure 3:

```
(THEREARE RULES (MENTIONS OBJCT PREMISE RECESSIONONTHEWAY) SET1)
```

It determines whether or not an object-level rule mentions the attribute *the economy may be heading for a recession.* Consider then the impact of second order meta-rules --- they can be used to select the criteria by which the KSs will be characterized. Thus we not only allow the user to specify refinement criteria, but make it possible for him to encode knowledge that decides dynamically which are the most appropriate criteria to use.

12.5 Content reference: limitations

There are a number of limitations and shortcomings in the mechanism described above, some of which arise from details of the current implementation, while others are deeper issues that are manifestations of more fundamental problems.

It is clear, for example, that the technique is not always applicable, for the simple reason that the relevant information may not be available in the KS code. Suppose for instance that some factor in the developmental history of the KS code were relevant. There is, for example, the well-known heuristic that code with recent, extensive changes is much more likely to have a bug than long established code with at most minor changes. Implementing this heuristic would require knowledge of the average length of time between edits or the percent of the code modified in the most recent edit. Yet such things could not typically be deduced from the code itself.

Another factor which can limit applicability of the content reference technique is the inherent complexity of some of the knowledge we might wish to encode. Given our current, fairly primitive ability to examine code, we are tempted to write KSs in a form simple enough that they can be examined by the rest of the system. Yet it would be unreasonably difficult to specify some more complex KSs in such a simple form. Our claim then is not that content reference is necessarily relevant in every case, but simply that where it is applicable, it offers a number of useful advantages.

The simplicity of content examining functions also presents problems. MENTIONS, for example, is simple in both its literal and "inferential" search capabilities. It does not yet, for instance, distinguish between appearance of the token in an affirmative rather than negative clause

and would be equally satisfied to find HIGH-RISK in a clause that said "the risk level is high risk", and one that said "the risk level is not high risk". In current use this has not been a problem, since the meta-rules written to date have been uncomplicated, but it is easy to imagine that a finer degree of distinction will in time be necessary.

This leads to the more general issue of the difficulty of examining code. This is important, since the utility of content reference and all our arguments about reliability depend on the ability to examine the code of a KS. Yet we have dealt somewhat blithely in previous sections with this difficult problem. There are at least two sources of difficulty. First, it can be difficult even for a programmer to examine an arbitrary chunk of code and understand it. Second, even if we could read it, it is not always clear how to analyze it: Try to specify for instance a procedure to determine whether one KS executes faster than another.

TEIRESIAS currently has only the simplest form of code examining ability, made possible by several useful shortcuts. First, the rules are viewed as a task-specific high level language. The primitive terms they use are both domain-specific, and reasonably abstract (e.g., possibility of a RECESSIONONTHEWAY, etc.). This makes their code much easier to decipher than, say, an assembly code version of the same thing. Second, the code is strongly stylized (the predicates and associative triples), allowing us to associate with each predicate function a template that helps decipher its code. Third, we can often gain much by using special case approximations of the desired characteristics of code. Consider the speed example above, for instance. Rigorous analysis of code is still an art much beyond our ability to automate. Yet in this case we can closely estimate the execution time of a rule by examining it to see whether any of its preconditions have yet to be examined, and hence will cause backchaining. If so, any rules which will be retrieved by that backchaining are similarly examined, etc. The amount of backchaining a rule will cause is then a close approximation to its likely execution time. Thus we need not solve the problem of analysis of algorithms exactly and for all representations, but can gain much by considering specialized approximations for the problem at hand.

We have thus far used our code examining ability primarily in some rather basic, syntactic ways. The "inferred mentions" is a step toward deriving more interesting semantic content by examining code, yet this is still clearly a difficult problem. But while the problem is difficult, it is also a separable issue. That is, the extent of the current capability to examine code is extremely elementary, but even the simplest form of it makes available the interesting capabilities displayed above. Thus, even though shortcuts like the template mechanism, etc., are not applicable to all forms of knowledge representation, wherever they can be applied some of the benefits of content reference become available.

The larger problem we're dealing with here is that of techniques for analyzing knowledge. It seems intuitively appealing to have the system reason about its knowledge base. But it is not clear whether this can be accomplished using concepts like "inferred mentions" or even the more general notion of content reference. Other approaches will have to be explored as well.

Another fundamental problem is the question of the uniformity of vocabulary of meta-level primitives at all levels. Will we be able to use the same set of primitives (like MENTIONS) at all levels, or will each higher level require additional primitives to use in referring to knowledge at the next lower level? If the latter, we have a larger and more difficult task to perform when building a basic set of primitives to provide user support.

We have also dealt blithely with the idea of replacing reference by name with reference by description. Yet it is not always clear how to generalize from a specific procedure to be invoked, to a general description of the capabilities desired. Consider the PLANNER fragments in Figure 5 for example and note the difficulty of replacing the "use" advice with more general characterizations.

There is also the issue of the amount of compute time required to do this form of

indirect referencing, compared to the speed of using names. Examining KS code to deduce some subtle property can be a time-consuming affair, especially when compared to the speed of reference by name. The potential for loss of speed becomes acute when we consider the time taken by inferring the presence of some more subtle property, which requires invocation of several rules for each object-level rule being examined.

However, as noted earlier, most of the computational cost can be paid in a background computation between performance runs. During that time, the system could compute the sets of KSs determined by the various descriptions. The results might either be saved for execution time use, or, in a form of "pre-compiling", the source code might be re-written, replacing the descriptions with the sets they define. The idea of computing such indexing lists before execution is of course a standard technique. We are simply suggesting that the basis for doing the indexing --- content reference --- might be more sophisticated than the simple literal-matching done in production rules, PLANNER's discrimination net, etc. By doing this, we obtain both the flexibility of reference by description, and the speed of reference by name.

We can even back off a little on our assumption about no new KSs being created during program execution. The worst that happens is that the original descriptions would have to be retrieved and applied to the new KS, to see if it meets their criteria. This could be done either "on demand" (as each reference criterion is about to be used), or the system might pause to "recompile". In either case, the time to accommodate a single new KS might not be unreasonable.

Finally, the dual use of object-level rules illustrated in Section 12.3 requires additional work. While the example given appears free of problems, it is not yet obvious that the transformation of rules from object-level to meta-level will always be as easy, nor that object-level knowledge will always be as totally appropriate at the meta-level as it was in this case.

12.6 Content reference and external descriptors

One apparent alternative to content reference would be formalizing the link between a set of external descriptors and the body of the KS. That is, our original formulation of the external descriptor approach said that the KSs were to be described *by the programmer* in terms of a given set of descriptors. By allowing the system itself to take over this task, we gain many of the benefits claimed above for content reference.

There are at least two ways this mapping might be provided. The first is a "descriptor verifier" approach. This would involve submitting to a deductive system both the code for a KS and some descriptors for it, then allowing the deductive system to attempt to prove the correctness of the descriptors (much like current work in program verification; also see [25] where this approach is explored further).

Another scheme would be to use a "descriptor generator", which, provided with the KS code and a characterization of the descriptors desired, would automatically generate them. A simple version of this was done some time ago: GPS was capable of constructing its own table of connections when supplied with operators in the form of rewrite rules (e.g., symbolic logic transformation rules) and a set of routines for defining differences [36]. It matched the left- and right-hand sides of the rules and applied each difference detector to the result. This was feasible because the KS "code" had a very simple form, and because the "characterization of descriptors" was a procedure for finding the relevant features.

But note that this still involves content reference: however we choose to implement the formalized external descriptor approach, we still require both access to and the ability to examine the body of a KS.

13 REASONING ABOUT INVOCATION CONTROL

One of the central contributions of the meta-rule framework is the suggestion that *programs can reason about control.* More specifically, in TEIRESIAS refinement of the set of possibly useful KSs is itself viewed as a problem solving task, with its own store of knowledge, its own heuristics, etc.

This is made possible by two key ideas. First, information about control has been separated out from object level knowledge, and has been represented explicitly (in meta-rules). Second, that information is accessible to the program itself -- the meta-rules can themselves be examined.

One additional idea contributes to the efficiency of our design: as noted earlier, there is a uniform representation of knowledge at all levels. In this case there are rules at all levels (but the use of rules is not crucial), making it possible to use a single inference engine (the same one used to run the object-level knowledge) to invoke all levels of knowledge.

The use of these ideas means that we have available at the meta level the full power of the original (object-level) inferential mechanism. The power of the basic mechanism is thus usable as an inferential and problem-solving tool in both the original task domain and in the domain of invocation control. This is what we mean by saying that TEIRESIAS reasons about control.[25]

Few previous systems have employed any of these ideas. PLANNER's use of several classes of theorems (antecedent, consequent, and erasing), for example, requires that the user decide how a chunk of knowledge is to be used before he can state the knowledge itself. The language thus requires that knowledge about a domain be combined with information about how to use that knowledge. Many of the systems reviewed above embed control information in the interpreter, rendering it inaccessible to the program. While PLANNER's recommendation list mechanism does put some of the information in the program rather than in the interpreter, it does not make explicit any of the underlying information about invocation control. The "use" construct, for instance, represents only the final result (the order of operators), without indicating the underlying reasoning that produced that order.

13.1 Motivation

Before examining how this view of refinement as problem-solving is implemented in the current system, it will be useful to consider *why* we might want to do it, since it may appear to be an extravagant use of computational resources. This reaction --- that it represents extravagant use of resources --- is predicated on the availability of the search vs. knowledge tradeoff. This tradeoff suggests that, in exploring a search space, one can either (a) search very quickly or (b) employ knowledge about the domain (e.g., meta-rules) to search more selectively. The tradeoff also suggests that one can search equally successfully by trading additional speed for less selectivity, or vice versa. But this is not always true. The tradeoff is sometimes quite unbalanced and sometimes not even available. There are at least three situations in which this can occur and in which we would prefer an intelligent choice rather than a fast choice.

First, we may prefer that selection be intelligent rather than fast in the situation in which a control structure saturates in the face of too many potential KSs. We can generalize this slightly and characterize it as any situation in which the potential computational costs of

---

25. In Section 12.3 we saw how the inference engine could be used to reason about the content of object-level rules. Here we are assuming the content has been determined and are using the inference engine to reason about the appropriate refinement for the set of rules.

unguided selection of KSs are large enough that it pays to invest resources in careful selection. This may arise simply because the number of potentially useful KSs is very large, or because the cost of invoking any one (even though there are few of them) is very large.

Recent work by Wilkins [53] demonstrates this very nicely. By devoting considerable computational effort to planning (rather than searching) and by guiding its planning through the use of a knowledge base about chess, his system is able to find appropriate moves as many as 19 ply deep. Yet in doing so it generates a search tree of only one hundred and nine nodes. Some of the best current chess programs, relying as they do on the search end of the tradeoff, would have to search approximately *10**14 nodes* to find the same move.[26] Clearly, in this case even a little knowledge can be more effective than a lot of additional speed.

Second, we may prefer that selection be intelligent rather than fast in situations where a KS can conceivably be invoked infinitely often. That is, each time it is invoked it produces *some* contribution, but we have no guarantee that it will either reach the desired goal eventually or ever become inapplicable. Theorem proving offers a good example. Consider a system that had (among others) the inference rule

```
HUMAN(X) ==> HUMAN(mother-of(X))
```

and the assertion

```
HUMAN(JOHN)
```

As Bundy illustrates in [4], we can have an arbitrary number of applications of the inference rule on this assertion. Each application will produce a new conclusion, so we know that the system is not simply looping. And it may be that some large number of applications of the rule are in fact necessary. Yet despite the system's non-looping, "productive" behavior, there is no guarantee of converging on a goal and no guarantee of eventual termination. Here even an arbitrary increment in speed will not substitute for more selective invocation.

The program cannot of course be allowed to loop endlessly this way, yet as noted in [4], neither is it appropriate to take the traditional approaches of outlawing all application of the rule (or more generally, all creation of new terms), or allowing a predetermined, fixed number of applications of the rule (allowing creation of a fixed number of new terms). In either of those cases, we would be unable to deduce the humanity of all of John's maternal parents more than N generations back (where N is the number of permitted applications of the rule). Instead we need some "intelligent" control on application of the rule, i.e., we require intelligent rather than speedy selection of KSs.[27]

Finally, intelligent selection of KSs can be important for systems which must change or interact with the real world in some way in order to explore possible alternatives, as for example in robot or computer-aided instruction systems. In CAI systems, for example, the intelligence of each decision by the program about how to structure the lesson and dialog, what questions to ask next, when to interrupt, etc., will have a significant impact on the observer's impression of its performance and utility. Unguided, exhaustive invocation (e.g., "ask all possibly relevant questions, in no particular order") is clearly inappropriate, even if computationally trivial. Yet a significant amount of computation can be devoted to choosing the next step "wisely" in only a few seconds of real time. A delay of this magnitude is often acceptable to the observer, and will in any case be preferred over a faster but less discerning choice of KSs.

---

26. Wilkins, personal communication; esimate based on an original branching factor of 32, which is then reduced to approximately 6 if we assume nearly optimal alpha-beta pruning.

27. The problem is apparently not uncommon: Rieger encountered much the same phenomenon in [39], and chose to cut off after an arbitrary number of inferences.

A similar situation arises in robot systems, or more generally, in "worldly" systems (those in which the results of problem solving and planning are acted on in the real world). In such systems we have to choose between alternatives because only one course of action can really be taken. This is often due to the impracticality of backtracking or "popping back to an earlier state" when possibly non-recoverable changes have been made to a real world environment. In that case, speed of invocation is almost irrelevant --- it matters little that we can quickly explore a certain course of action if that action is likely to lead not only to failure to solve the problem, but to a world state in which the problem is now more difficult to solve.

Thus, given in some situations the unevenness (or lack of) the search vs. knowledge tradeoff, the importance of an informed choice becomes clear.

One final point. We have chosen to make that informed choice by using a fairly sizable mechanism. We have suggested making explicit in the program the process by which a set of KSs is refined and suggested effecting that refinement using the full inferential and problem solving mechanism available at the object level. Such an investment of effort is useful as long as the refinement process needs to be carried out often, as for example, when the appropriate ordering and pruning of the set varies considerably with variations in the problem state. Otherwise a less complex mechanism (e.g., an order pre-set by hand) may be adequate. This simply reflects the conventional wisdom that small problems that can be solved once are often easier done by hand, while larger problems that are encountered repeatedly in varying form are often worth the investment of writing a program.

13.2 Implementation

In saying that meta-rules "reason" about control, we mean more specifically that (like object-level rules) they have a cumulative effect in which several different perspectives may be considered.[28] Unlike, say PLANNER's "use" or theorem base filter mechanism, with meta-rules the final decision on invocation is the result of combining contributions from several sources of information. The central point is that the knowledge about invocation control has been embodied in a number of (what are designed to be) independent judgments, each providing its own source of evidence about appropriate use of object-level knowledge. These distinct (and perhaps conflicting) perspectives on relative utility or ordering of object-level knowledge are combined to arrive at a final decision.

Meta-rules also reason in the sense that, like object-level rules, they may require an inference chain several steps long. In the example in Section 12.3, for instance, three inference steps were required to infer the presence of the concept HIGH-RISK.

There are a number of other ideas in the meta-rule framework which are used to make possible reasoning about control and which turn out to offer other useful capabilities. First, having decided to reason about invocation control, it makes sense to separate out and specify clearly the knowledge used in doing that reasoning. Simply by writing down the actual decision criteria (in this case in the form of meta-rules, but the representation is not critical), we are making the invocation control criteria explicit, as opposed to implicit mechanisms like QA4's GOALCLASS and PLANNER's THUSE. One result of this is that we have, as noted in Section 11.2.1, explicit representation of invocation control information.

---

28. Note that this is independent of the judgmental aspects of the rules and would be true even if meta-rules used standard Boolean logic. In that case we might simply count the "votes" of meta-rules for and against utility of a given object-level KS, and order the set of plausibly useful KSs accordingly. As noted earlier, the nature of the evidence combining function is a non-trivial question. We have suggested a simple majority function here for purposes of illustration and to emphasize that, however they are to be combined, the use of multiple sources of knowledge is the important issue.

Second, because those criteria are embodied in reasonably small, stylized "chunks" which the system can access, dissect, and examine, the invocation control criteria become program-level objects rather than inaccessible information embedded in, say, the system interpreter (as is done in traditional production rule systems, for example).

Finally, this approach facilitates the construction of a base of knowledge about invocation control. Just as it proved useful at the object-level to separate out the (object-level) knowledge from the inference engine, so it seems appropriate as well at the meta-level to separate out from the inference engine a second knowledge base of information about invocation control.

13.3 The utility of reasoning about invocation control

The utility of reasoning about invocation control lies, as noted earlier, in the power it can supply in guiding problem solving performance. There are as well advantages associated with the particular approach taken here. For example, the idea of embodying the invocation control criteria in a form which the system can both access and dissect means that the criteria are at once both inference rules and data. The same object in the program is at once a mechanism that guides the use of knowledge and a data structure which is in turn examined by the criteria at the next higher level.

The explicit representation of information about control embodied in meta-rules also makes it possible for the system to reason about that control information. This is a specific example of a more general idea which has seen several instantiations in recent AI work. The more general concept, simply stated, is that explicit representation of knowledge about a topic enables a program to reason about that topic, often producing useful new capabilities. In [41], for example, the structure of a plan under consideration was available to the program for examination and manipulation, and this made possible interesting advances in planning and problem solving. In [49] and [15], interdependencies of assertions were represented and manipulated by the program, making possible an interesting form of backtracking and a non-monotonic logic system. In [12], the architecture of the data structures used by the system was represented explicitly, making possible a number of useful knowledge acquisition and knowledge base maintenance techniques. In [33], a task network provided the foundation for the system's ability to reason about and reformulate its current set of tasks. Similarly, we have seen here reasoning about control made possible by explicit representation of control information.

The explicit encoding of invocation control criteria, combined with the uniformity of representation and uniformity of inferential mechanism noted earlier means a very easy extension of the "reasoning about control" facility to all levels. Just as the benefits of using heuristics to guide heuristics applies at all levels of the framework, and the notion of content reference is useful at all levels, so our ability to reason about control extends easily to all levels, with an appealing economy of machinery. The same inference engine which solves (stock market) problems on the object level is used to solve problems (of knowledge source selection) on all the meta-levels.

In dealing with problems of knowledge source selection, the inference engine would be using knowledge about control, which, as noted above, could be stored in its own knowledge base. There are advantages in having a separate knowledge base, rather than embedding information in the inference engine. First, a separate knowledge based is in general easier to build, since it often permits incremental changes. In this case that means our store of knowledge about invocation control can grow incrementally. Keeping the information in a separate knowledge base also makes it far more "substitutable" than it might be if it were embedded in the inference engine. This facilitates the replacement of one body of knowledge with another for purposes of testing or comparison. In the current context, this means it is relatively easy to try out different bodies of strategy knowledge. Finally, a separate knowledge base increases the

transparency of the system, since it makes accessible the information on which system behavior is based.

13.4 Reasoning about invocation: limitations

Our approach to reasoning about invocation control has a number of limitations arising from the level of power and user support it provides, and from the scope it encompasses.

Our basic approach -- which suggested that the inference mechanism used for problem solving at the object-level can as well be applied to reasoning about invocation control at the meta-levels -- implies that our ability to reason about invocation control will inherit both the power and the limitations of the basic inference mechanism. The forms of knowledge captured well at the object-level will similarly be easy to express at the meta-level; those difficult to represent at the object-level will prove equally difficult at the meta-level. The relevant question is then, Does knowledge about controlling invocation differ significantly in representational requirements from the task-specific knowledge of more traditional applications? Or is it in fact "just another domain" whose relevant concepts are representable with the range of techniques currently available? Our early experience suggests the latter, but the issue is still an open question and more experience is required.

Another potential weakness inherited from the object-level inference mechanism is the methodology of dealing with conflicting evidence. As noted previously, it is not clear whether all "differences of opinion" (whether at the object- or meta-level) can be reduced to numeric terms and settled via some numeric comparison. Our current implementation inherits from the object-level the assumption that numeric comparisons can indeed be made, but this is not necessarily going to prove true in the long run.

Alternative formulations have been explored in other systems. The NASL system employed a three-phase *rule-in*, *rule-out*, *reformulate* cycle. The first phase attempted to find a reason why a particular KS was especially good, the second tried to rule out KSs, and the third tried to reformulate the current goals by synthesizing them into a new goal. If at any time this process selected out a single KS, the cycling stopped and that KS is invoked. A range of other refinement paradigms are no doubt possible; this is an interesting focus for future work.

There is a potential weakness, too, in proposing a framework for knowledge organization and use, rather than proposing a specific solution to a particular task. There remains for the user the substantial burden of instantiating the framework for his own application. We have suggested a way of thinking about, organizing, and using information about controlling invocation. We have not suggested what that information might actually be. The task of specifying the information is a sizable one, whose difficulty is one measure of utility of the framework.

Once the framework has been instantiated there is also the question of effectiveness. In terms of direct experience, we have as yet only the brief empirical evidence of the system described above to suggest that this approach of building a knowledge base about invocation control will provide sufficient power. There is indirect evidence, however, in the widespread use of knowledge-based systems in a range of task domains. Assuming that the representational requirements are similar enough, the success of other knowledge-based systems suggests that the approach may prove profitable. If a large store of task-specific knowledge is effective at the object level, it may well prove equally effective at the meta-level.

One practical difficulty lies in our lack of extensive experience with the meta-rule framework. As noted earlier, it has been applied in the medical domain (the MYCIN system) and simulated in detail for the investment domain. This has been sufficient to suggest the basic utility of the technique, but has not provided sufficient experience to gauge such factors as the utility of some of the higher levels of meta-rules. How often, for instance, will we encounter domains in which we can specify second or third order rules? We require additional

experience in new domains before we will be able to evaluate the practical utility of all facets of the technique.

14 APPROPRIATE PROBLEMS

We have explored here a particular perspective on guiding invocation and examined a number of mechanisms based on it. We have noted in passing several caveats about recognizing classes of problems for which its use is appropriate. It may prove useful to review them briefly here.

It must first of all be possible to decompose the problem into parts of roughly the right size. If the problem doesn't decompose, then we have no set of KSs to choose from; if it is decomposed into parts that are too small, we may be hard-pressed to produce non-trivial descriptions of them.

There must also be, as noted, some degree of "ill structuredness" or indeterminacy to the problem. Our approach seems to make little sense if the problem can be broken down into a totally specified sequence of operations, with no uncertainty about what to do next.

We require too, some minimal size to the problem. Where there is only minor indeterminacy and a relatively stable collection of KSs, the overhead of the techniques we have suggested may not be worth the level of performance they might add. It is useful as well if we are working with a system which has not only a large number of KSs, but a large number of them applicable at any one point. This makes the need for refinement more compelling.

Our techniques can also be useful when building large systems subject to frequent developmental changes. Recall the discussion in Section 12.4.1, which demonstrated that using content reference made the system in part self-adjusting: changes to KSs and invocation criteria were automatically accommodated. This can be a useful property as the number of KSs in the system gets large, and the burden of reviewing them begins to be unmanageable. But the changes must be frequent enough to be worth the overhead our technique requires. Recall the resolution theorem proving example, which illustrated a domain with infrequent enough changes that relatively little benefit was derived from additivity.

There is some question, too, of whether, even for a sufficiently large problem, all of the mechanism and machinery we have outlined is always appropriate. Note, however, that the three basic concepts we have discussed are all quite separable. We suggested first, that a certain style of invocation control (manipulation of the set of plausibly useful KSs) appears to be both a reasonably powerful and fairly natural way of encoding strategy information. While there are many ways to implement such control, we suggested next that one particular reference technique (content reference) offers a number of advantages, including flexibility of the resulting system. Finally, we indicated that the mechanism used to arrive at the appropriate reordering might usefully be a general problem-solving and inference mechanism, and more specifically, the same inference mechanism in use at the object level.

If system flexibility and the capability to reason about KS use are not primary concerns, then one may still usefully adopt our view of invocation control. If flexibility is a concern, then content reference may be useful as well, and so forth. We can thus employ as few or as many of these ideas as is deemed appropriate, offering some degree of variability in adopting the framework suggested here.

In general, our view appears well suited to problems which can be appropriately decomposed, and for which there is some degree of ill-structuredness, arising either out of the nature of the task-domain, or the problems of incremental construction of large systems.

15 APPLICATIONS TO RETRIEVAL

Two of the themes discussed in previous sections dealt with the application of a general inference mechanism and content reference to the refinement phase of invocation. We saw how these ideas can provide a number of useful capabilities. Recent work suggests that these ideas can equally well be applied to the retrieval phase. Since we have only begun to explore this view, we deal with it here only briefly, to suggest the future direction of this line of work.

To see the utility of this application, recall that we view invocation as a three step process: retrieval, refinement, and execution. Of the three phases the first has traditionally received the most attention. It determines which KSs will be considered for execution and is typically equated with the "kind" of invocation used. For example, goal-directed invocation is retrieval based on the goal a KS can achieve, data-directed invocation is retrieval based on the input data used by the KS, while means-ends analysis is retrieval based on the difference to which the KS is applicable (and hence might be called "difference-directed invocation"). Both a general inference mechanism and content reference can be very useful at this first stage of the invocation process.

15.1 General inference mechanism

In exploring the application of a general inference mechanism to the refinement phase, we suggested that we could view control of invocation (refinement) as problem solving. Thus, rather than using a simple, predetermined ordering of KSs (e.g., PLANNER's THUSE or QA4's GOALCLASS), we instead employed a more general inference mechanism that allowed the system to deduce the appropriate guidance.

We might, in analogous fashion, apply such a mechanism to retrieval. Rather than having a single, pre-defined retrieval criterion (as in a program designed to do strictly goal-directed invocation), we might instead give the program an inference mechanism which would allow it to deduce the best retrieval criterion to use (i.e., deduce the best control regime). With this approach, we are viewing *retrieval* as problem-solving: The decision about which retrieval criterion to use is seen as a problem-solving task, one which would have its own store of knowledge and its own body of heuristics and strategies.

This might be implemented by having a knowledge base of, say, rules for choosing the appropriate retrieval criterion, and allowing that choice to be made several times over the course of a problem solution. This could result in a system with the ability to switch its control structure dynamically as a problem progresses.

Consider, for example, a program designed to do heuristic search. For domains in which a single search procedure does not provide effective performance, it would prove very useful for the program to be able to decide from moment to moment what form of search would most likely be successful. It might thus use forward search at one point, backward search at another, means-ends analysis at another point, and so on. Note that we are not speaking of a pre-programmed succession of techniques, but of an ability to choose each technique in response to changes in the problem state.

Two simple rules for the heuristic search domain would be:

```
If    there are numerous paths leading from the initial state, and
      there are very few paths entering the final state,
then  backward chaining is likely (.8) to be useful.
```

and conversely

```
If    there are very few paths leading from the initial state, and
      there are numerous paths entering the final state,
then  forward chaining is likely (.8) to be useful.
```

A real system, of course, would have to confront more difficult (and interesting) distinctions. What are all the properties of the search space that would allow us to choose between, say, depth first, breadth first, best first, and hill climbing? What changes in our ability to estimate costs would lead us to change say, from best first to beam search? We are currently developing rules which attempt to capture this information.

Note the change in our use of the general inference mechanism. Where, earlier in the paper, the general inference mechanism was used to deduce the appropriate ordering of KSs already retrieved according to a predetermined criterion, here the mechanism is being used to select the retrieval criterion (and hence the control regime) itself. We are currently exploring the benefits this technique may offer for developing more powerful problem solving systems.

As in the evolution noted earlier from the British Museum algorithm to means-ends analysis, the selection of a good retrieval criteria appears to offer great potential for control of problem solving. We speculate that, as careful refinement can offer a significant improvement over exhaustive or unguided invocation, so dynamic selection of appropriate retrieval may be an even greater improvement over "static" retrieval.

15.2 Content reference

The idea of referring to KSs by describing them, and effecting this description by content reference also has utility at the level of retrieval criteria selection. As before, it makes possible a system with greater flexibility and reliability. To see this, consider the performance program described in Section 6. In that system goal-directed invocation is implemented via predefined lists of rule names (as in Figure 7) which were originally constructed by hand. Because this is reference by name, it is neither flexible nor reliable (in the sense defined earlier). Arbitrary changes can be made to the body of these rules, yet they will be retrieved no differently after the changes than before. But if the retrieval (and hence invocation) were instead based on content reference ("use rules which mention in their conclusion which stock to invest in"), retrieval would automatically reflect any changes in rule content.[29]

We also noted earlier that content reference makes it easier for the programmer to define new ways of referring to KSs, and explored the utility of this in the refinement phase. It is equally useful in the retrieval phase. We could, for instance, define somewhat more complex control regimes like bi-directional invocation ("use knowledge sources that mention either the current goal or the current conclusion..."), or "one-step" invocation ("use KSs that mention both the current goal and the current conclusion..."). More esoteric regimes are possible (speed-directed, side-effect-directed), if only it proves possible to produce functional definitions of them (which of course may not be an easy problem).

When content reference is used in the retrieval phase of invocation, we refer to it as *content-directed invocation*, to suggest its place in the ongoing development of invocation techniques. Invocation schemes explored to date have included absolute pointers (procedure calling), symbolic reference (procedures as parameters in ALGOL), pattern-directed invocation (production rules, PLANNER), and finally content-directed invocation. Note also that there has been a parallel evolution in access to memory locations, from absolute binary, to symbolic

---

29. While the rule lists in MYCIN were originally constructed by hand, this was replaced some time ago with a simple form of content reference. This automates the list construction process and provides a degree of reliability and flexibility.

addressing in assemblers, to relocatable core images, on up to content addressable memories (e.g., LEAP [18], or PLANNER's pattern matching). Content-directed invocation can thus be seen as the procedural analogue of content addressable memory.

It is interesting to put this idea in historical perspective, and consider that some of the developments in programming languages can be viewed as attempts to be more descriptive about which KS to invoke. The programmer using procedures effectively says "give me that KS next", indicating it by name. In GPS, STRIPS, and traditional production systems, the user has little or no control over which KS is invoked next. PLANNER make it possible to say "give me any KS whose pattern matches this one"; in using that pattern as a designator of a goal, the request becomes "give me any KS that achieves the goal designated." Content reference makes it possible to say "give me any KS that fits the following description", offering the potential for a wide range of KS retrieval (i.e., invocation) schemes.

16 CONCLUSIONS

We have defined the notion of saturation and argued for its inevitability on the basis of both the inherent non-determinism of AI problems and the apparent necessity of large stores of knowledge to provide problem-solving power. In that context we viewed the concept of a strategy as advice about which of the several KSs retrieved ought to be used next and considered it more generally as one form of meta-level knowledge.

We explored an approach to dealing with saturation based on a mechanism for pruning and reordering the set of plausibly useful KSs. This approach was seen to have a number of advantages associated with it, including making possible comparative statements about which KS to use and the oppoitunity to encode fairly specific strategies without causing an unreasonable amount of overhead at execution time. Our approach also offered an interesting degree of uniformity of representation, which in turn makes possible a hierarchy of strategies and enables us to apply at the meta level the same explanation capabilities developed for object-level knowledge.

Meta-rules also provide a means for representing invocation control information as an explicit, program-level construct. Explicit representation of this information makes for a system which is more transparent, less prone to what we termed partial order bugs, and easier to modify. Representing the information as a program-level construct (rather than embedding it in the code of an interpreter) allows us to apply the basic concept of strategy at all levels of the hierarchy: Strategy knowledge at one level is data to the strategy at the next higher level.

We described a technique called content reference that is used by meta-rules and showed that it offered a useful degree of flexibility. In particular, a number of standard kinds of modifications to a knowledge base can be effected with little or no need for additional bookkeeping. We also showed that content reference can be a reasonably complex process involving inferences about the content of a KS, rather than just simple pattern-matching.

We explored the utility of giving a program a general inference mechanism with which to deduce the appropriate strategy to use. This was seen to be more flexible than predetermined lists of KSs and led to the suggestion that we can view control of invocation as problem-solving. The decision about how to reorder or prune the list of KSs retrieved can itself be viewed as a problem-solving task, with its own heuristics and body of knowledge. It is in this sense that TEIRESIAS reasons about control.

While the machinery we have described may appear somewhat elaborate, there is in fact an interesting degree of economy in much of our design. There is an economy of application, an economy of machinery, and an economy of knowledge representation. By economy of application, we mean that the ideas we have suggested are all uniformly applicable across the

multiple levels of the meta-rule hierarchy. The notions of invocation control via pruning and reordering, the use of content reference, the use of a general inference mechanism, and the explicit representation of strategy knowledge are all equally useful at all levels, and can be applied at all levels without modification of the basic concept.

By economy of machinery, we mean that the mechanisms used to implement these ideas need be constructed only once and can be applied in several different ways. The inference mechanism used at the meta level to solve the problem of selecting the appropriate strategy, for instance, is the same inference mechanism used at the object level to solve problems of investment selection. The same inference mechanism also forms an important part of the content reference machinery, where it is used to infer the presence of concepts which have not been mentioned explicitly in a rule. The same mechanism is thus used to reason about strategy selection, about investment choices, and about the content of rules in the system.

The use of the same interpreter at both the object and meta level has as well a certain aesthetic appeal. While it does not appear to offer any new problem solving power, it seems somehow intuitively appropriate, a feeling reinforced by criticism other system designers have made about their own work when it lacked such uniformity (see, e.g., [33]).

By economy of knowledge representation, we mean that the idea of meta-level knowledge allows us to make multiple uses of the same body of knowledge. Rules at one level in the hierarchy, for instance, are used to control the invocation of rules at the next lower level, while at the same time they are "data" to the rules above, which examine and reason about them. We also saw that the same body of knowledge (e.g., rules about HIGH-RISK) can be both object-level knowledge used to reason about the domain, and meta-level knowledge used to reason about rules about the domain.

We have thus taken a fairly small set of ideas and have found numerous applications for those ideas within a single framework. The result is a fairly elaborate collection of machinery, but one which is not inherently complex.

One final comment. One of the interesting side effects of work on applications programs like MYCIN, DENDRAL [3] and MACSYMA [30] has been the formalization of knowledge about their respective fields. In some cases this has also led to new insights in these fields. The view explored here suggests treating both strategies (the refinement phase of invocation) and control regimes (the retrieval phase) as "just another application domain", and building for each a knowledge base of information and heuristics. One potential (though surely long-term) result of this line of work is a similar side-effect: the formalization and possible extension of our knowledge about strategies and invocation techniques.

Acknowledgments

Bruce Buchanan provided valuable assistance and encouragement during the development of the ideas presented above. Jim Davidson, Drew McDermott, Jon Doyle, Lee Erman, Carl Hewitt, Gerry Sussman, and Peter Szolovits offered useful comments on earlier drafts of this paper. My financial advisor Malcolm Jones displayed great patience in explaining to me intricacies of economics and the stock market. The small set of rules shown above is in no way representative of the wealth of information he supplied. In a few more years perhaps we shall be able to report on the wealth his information supplied.

References

[1] Adams J B, A probability model of medical reasoning and the MYCIN model, *Math Biosciences*, 32:177-186 (1976).

[2] Bobrow D, Winograd T, An overview of KRL, *Cognitive Science*, vol 1, pp 3-47, January 1977.

[3] Buchanan B G, Feigenbaum E A, DENDRAL and Meta-DENDRAL: Their applications dimension, *Artificial Intelligence*, 11, August 1978, pp 5-24.

[4] Bundy A, Exploiting the properties of functions to control search, D.A.I. Research Report No. 45, University of Edinburgh, October 1977.

[5] Bundy A, et. al, Solving mechanics problems using meta-level inference, *Proc 6th IJCAI*, pp. 1017-1027, August 1979.

[6] Chang C-L, Lee C-T, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.

[7] Davis R., Generalized procedure calling and content-directed invocation, *SIGART*, No. 64, August 1977, pp 45-54.

[8] Davis R, Applications of meta level knowledge to the construction, maintenance, and use of large knowledge bases, in Davis and Lenat, *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, in press.

[9] Davis R, King, J J. An overview of production systems, in *MI 8: Machine Representations of Knowledge*, (Elcock and Michie, eds), John Wiley, 1977, pp 300-334.

[10] Davis R, Buchanan B G, Shortliffe E H, Production rules as a representation for a knowledge-based consultation system, *Artificial Intelligence*, 8:15-45, Spring 1977.

[11] Davis R, Buchanan B G, Meta-level knowledge: overview and applications, *Proc 5th IJCAI*, Aug 1977, pp 920-927.

[12] Davis R, Knowledge about representations as a basis for system construction and maintenance, in *Pattern-Directed Inference Systems*, (Waterman and Hayes-Roth, eds.), Academic Pres, 1978, pp 99-134.

[13] Davis R, Interactive transfer of expertise: acquisition of inference rules, *Artificial Intelligence*, 12 (1979), 121-157.

[14] de Kleer J, et. ai, AMORD: Explicit control of reasoning, *Proc. ACM Symp. on Artificial Intelligence and Programming Languages, SIGART/SIGPLAN* combined issue, August 1977, pp 116-125.

[15] Doyle, J. A truth maintenance system, MIT AI Memo 521, April 1979.

[16] Erman L, Lesser V, A Retrospective View of the HEARSAY-II Architecture, *Proc Fifth Intl Joint Conf on AI*, Aug 1977, pp. 790-800.

[17] Ernst G W, Newell A, *GPS: A Case Study in Generality and Problem Solving*, Academic Press, New York, 1969.

[18] Feldman J, et. al, Recent developments in SAIL, an ALGOL-based language for artificial intelligence, Stanford AI Memo 176, November 1972.

[19] Fikes R, Nilsson N, STRIPS: a new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, 2:189-208, Winter 1971.

[20] Green C C, The application of theorem proving to question answering systems, Stanford AI Memo 96, August 1969.

[21] Hayes P J, Computation and deduction, *Proc 1973 Math. Foundations of Computer Science Symposium*, Czech. Acad. Sci.

[22] Hayes P J, In defence of logic, *Proc 5th IJCAI*, August 1977, pp 559-565.

[23] Hayes-Roth F, Lesser V R, Focus of attention in the HEARSAY II speech understanding system, Carnegie-Mellon Computer Science Dept Report, January 1977.

[24] Hewitt C, Description and theoretical analysis of PLANNER, PhD Thesis, Department of Mathematics, MIT, 1972.

[25] Hewitt C, Procedural semantics: models of procedures and the teaching of procedures, in Rustin (ed.), *Natural Language Processing*, Courant Computer Science Symposium Series, vol 8, 1974.

[26] Kornfeld W, ETHER -- a parallel problem solving system, *Proc 6th IJCAI*, Tokyo, Japan, 1979, pp. 490-492.

[27] Kowalski R, Algorithm = logic + control, *CACM*, 22:424-436, July 1979.

[28] Lenat D B, AM: An AI approach to discovery in mathematics, in Davis and Lenat, *Knowledge-Based Systems in AI*, McGraw-Hill, in press.

[29] Lesser V R, Corkhill D, The application of artificial intelligence techniques to cooperative distributed problem solving, Technical Report, Comp and Inf Science Department, Univ of Mass., February 1979.

[30] Mathlab Group, The MACSYMA reference manual, Version 9, MIT Laboratory for Computer Science, December 1977.

[31] McCarthy J, Programs with common sense, in Minsky, (ed), *Semantic Information Processing*, MIT Press, pp 403-418.

[32] McDermott D V, Sussman G J, The CONNIVER reference manual, MIT AI Memo 259a, MIT, January 1974.

[33] McDermott D V, Planning and acting, *Cognitive Science*, 2:71-109(1978).

[34] McDermott D V, Vocabularies for problem solver state descriptions, *Proc 5th IJCAI*, pp

229-234, August 1977.

[35] Newell A, Simon H A, *Human Problem Solving*, Prentice Hall, 1972.

[36] Newell A, Shaw J C, Simon H A, A variety of intelligent learning in a general problem-solver, in Yovitts and Cameron (Eds.), *Self-organizing Systems*, Pergamon, New York, pp. 153-189.

[37] Newell A, Heuristic programming: ill-structured problems, in *Progress in Operations Research*, vol 3, John Wiley & Sons, 1969.

[38] Pratt V R, The competence/performance dichotomy in programming, *4th ACM Symposium on Principles of Programming Languages*, Jan 1977, pp 194-200.

[39] Rieger C J, Conceptual memory, AI Memo 233, Stanford AI Lab, July 1974.

[40] Rulifson J F, et. al., QA4: A procedural calculus for intuitive reasoning, Stanford Research Institute Technical Note 73, November 1972.

[41] Sacerdoti E, *A structure for plans and behavior*, American Elsevier, 1976.

[42] Scott A C, Clancey W, Davis R, Shortliffe E, Explanation capabilities of rule-based consultation systems, *Am Jnl Computational Linguistics*, microfiche 42, 1976.

[43] Shortliffe E H, Buchanan B G, A model of inexact reasoning in medicine, *Mathematical Biosciences* 23 (1975) pp 351-379.

[44] Shortliffe E H, *Computer-based Medical Consultations: MYCIN*, American Elsevier, 1976.

[45] Simon H A, The structure of ill-structured problems, *Artificial Intelligence*, 4(1973), 181-201.

[46] Smith R G, Davis R, Distributed problem solving: the contract net approach, *Proc 2nd Natnl Conf of CSCSI*, Toronto, July 1978, pp 278-287.

[47] Sussman G J, Winograd T, Charniak E, microPLANNER reference manual, MIT AI Memo 203a, December 1971.

[48] Sussman G J, McDermott D V, From PLANNER to CONNIVER -- a genetic approach, *Proc Fall Joint Computer Conference*, 1972, pp 1171-1179.

[49] Stallman R M, Sussman G J, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit design, *Artificial Intelligence*, October, 1977, 9:135-196.

[50] Waldinger R, Levitt K N, Reasoning about programs, *Artificial Intelligence*, 5:235-316, Fall 1974.

[51] Warshall S, A theorem on Boolean matrices, *JACM*, 9:11-12, January 1962.

[52] Weyhrauch R, Prolegomena to the theory of formal reasoning, AI Memo 315, Stanford AI Lab, December 1978.

[53] Wilkins D, Using plans in chess, Proc 6th IJCAI, Tokyo, Japan, 1979, pp. 960-967.